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 |
| v1.4 | 10/29/2015 | Manuel Gonzalez | Added details to the High Level Architecture section |

**Team Members:**

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

**Contents of this Document**

High-Level Architecture

Design Issues

1. **High Level Architecture**
   1. **Major Components**

The Jarvis Emulator will be composed of 7 modules that will be implementing Microsoft .NET Observer Pattern to interact between them. This is an example of a Publish-Subscribe style, 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.

Our system also integrates some ideas of the Pipes-and-Filter style. The configuration module acts as a filter for the speech recognized data, transforming the recognized command into an action that depends on the current user’s configuration. Again, this increases modularization because the speech recognizer does not have to hold the user’s information to be able to parse the spoken commands.

The 7 modules are as follows:

* + 1. Face Recognizer Module: Takes input from the camera and detects the current user using face recognition algorithms.
    2. Speech Recognizer Module: Takes input from the microphone and detects any spoken command said in the room.
    3. Configuration Module: Holds and manages all the information related to each user, such as their name or frequented websites or programs.
    4. Actions Module: Responds to any request and executes some system actions or routes the request to the corresponding modules.
    5. RSS Feed Module: Gathers data from the internet based on the given URL (which has to point to the RSS feed of the website).
    6. UI Module: Handles the input from the user in order to update the information hold by the configuration module and outputs some user notifications.
    7. Speech Constructor Module: Outputs the requested data in natural language format and some user notifications through the speakers.
  1. **System Interfaces**
     1. **User - System:**
        1. Camera: Input needed to be able to detect the user through face recognition
        2. Microphone: Input needed to be able to detect spoken commands
        3. Display: Outputs the UI elements
        4. Speakers: Outputs the audio used to notify the user verbally
     2. **Internet - System**
        1. HTTP: Protocol used to get data from the internet
     3. **Module - Module**
        1. Active user id message: Holds which user is in front of the camera
        2. Command, keyword pair message: Holds which command was parsed from the microphone as well as some information to be filtered by the Configuration Module
        3. URL message: Holds the URL address pointing to the website requested
        4. Filepath message: Holds the address pointing to the program requested
        5. Error message: Holds an error message that occurred somewhere on the system
        6. Parsed Data message: Holds the parsed data gathered from the internet
        7. Change Configuration message: Will cause the configuration module to change its state
        8. User notification message: Holds a message to be output to the user

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. Take into account that the Actions Module is listening to the errors of the rest of the modules, but those interactions were not included to reduce the diagram’s complexity.



**Design Issues**

1. **Design Trade-off**
2. 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.
3. **Prototyping**
4. 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.
5. **Technical Difficulties and Solutions**
6. 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.
7. 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.
8. **Maintainability**
9. 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.
10. **Reusability**
11. 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.
12. **Testability**
13. 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.
14. **Risks**
15. 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.