Jarvis Emulator  
High Level Design  
COP 4331, Fall 2015

**Modification History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Who** | **Comment** |
| v0.0 | 10/26/2015 | Robin Schiro | Created document |
| v1.0 | 10/27/2015 | Jimmy Lam | Added Design Issues |
| v2.0 | 10/28/2015 | Manuel Gonzalez | Added High Level Architecture |

**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**

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:

* + 1. Face Recognizer Module
    2. Speech Recognizer Module
    3. Configuration Module
    4. Actions Module
    5. RSS Feed Module
    6. UI Module
    7. Speech Constructor Module
  1. **System Interfaces**
     1. **User - System:**
        1. Camera
        2. Microphone
        3. Display
        4. Speakers
     2. **Internet - System**
        1. HTTP
     3. **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.



1. **Design Issues**
2. **Design Trade-off**
3. 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.
4. **Prototyping**
5. 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 an action module.
6. **Technical Difficulties and Solutions**
7. 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.
8. 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.
9. **Maintainability**
10. Using the publish-subscribe design, for the project to run as a whole, each module communicates through the action module. For instance if the user calls for weather updates, the speech recognition will parse the user’s voice, and, through the action module, call for the RSS feeds from a weather website, and these feeds will be sent 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.
11. **Reusability**
12. With a high decoupled design, each of our modules can function dependently of one another, and so each module can be reused in other programs, such as those requiring facial recognition, RSS feeds, speech construction or recognition.
13. **Testability**
14. Our project has a high decoupled design, and 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 right subscriber, making the testing process highly involved.
15. **Risks**
16. Using a publish-subscribe architecture, we may encounter issues in the future, if we decide to add new modules. Since the modules are subscribed to the user’s speech and the action module, we must make sure that future modules are subscribed to the user and the action module (whichever that applies), and update the action module to be able to call the added module so that Jarvis functions properly.