RO2

FYP Final Report

Wearable Computing Based on Voice Control

by

Wai Man Chan

RO2

Advised by

Prof. David ROSSITER

Submitted in partial fulfillment

of the requirements for COMP 4982

in the

Department of Computer Science

The Hong Kong University of Science and Technology

2014-2015

Date of submission: April 21, 2015

Abstract

Voice User Interface and wearable are popular topics in these few years. Current approach requires wearable devices to equip a display for Graphic User Interface, and popular voice assistances require a low latency Internet connection to make it usable.

This project explores the possibility of extending the traditional voice command approach, to eliminate the need of Internet connection, and voice notification, for Voice User Interface, which enables future wearable device for a better user experience, and allows third party extension.

1. Introduction

1.1 Overview

With a shipment forecast of 275 million units in 2019 [1], wearable computers are the “next best thing” in the technology sector. These are miniature electronic devices that are suitable to wear on the body or clothing. They are superior compared to the current smartphone and tablet technology, in the sense that they deliver consistent interaction between the device and the user. The user does not even need to turn on or pick up the device [2].

The current mainstream implementations of wearable computing are smart wristbands (activity trackers) and smart watches, which can notify the users of notifications from their phone and measure users’ vital statistics [3][4]. Smart watches feature a touch-supported display, which allows users to read text and consume information on a graphical user interface (GUI). However smart watches lack a lengthy battery life to be considered useful for normal daily life.

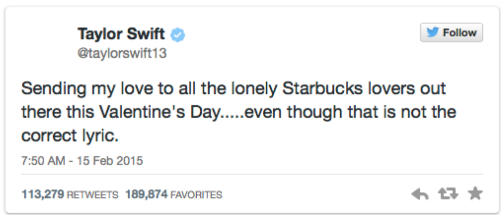
Several companies have released smart glasses, which mount a transparent color display on one side of the glasses. One of the famous examples is Google Glass, which has still not been released into the consumer market [5]. However, it has already faced a lot of resistance in its adoption by its early testers, including usability and legal challenges.

The only components that differentiate the hardware of wearable devices from any other are the remote sensors and vibration motors. The software interface itself is still identical to other devices and falls under the umbrella of GUI, a technology invented by researchers at Xerox PARC in 1973, and popularized by Apple in 1984 [6]. In the next 30 years, while the color pattern and logos have varied between devices, the structure of GUIs have not changed radically, ranging from window/card-based areas to display information and widgets for reading and editing information. After three decades of development, GUI have become the most popular user interface, as they are included in more than 95% of desktop computers [7]. However, to adopt GUI, wearable devices have to include a graphic processing unit to render the interface, and a display panel to display the interface. Although technology companies have leverage variety of display technologies in making wearable device, including AMOLED displays [8], and monitoring user motion to automatically switch on the display [9], display panel still consumes a lot of energy, which shortens of the time between recharges dramatically, and produces a poor user experience.

Voice User Interfaces (VUIs) are the alternative to GUI. VUIs are defined as computer interfaces, which leverage speech synthesis and recognition technology to interact with the user in a normal conversation [10].

A number of attempts have been made throughout the history of computers to achieve a workable VUI, starting which Apple, which tried to make use of the digital signal processor (DSP) inside Macintoshes in 1993. Later, Microsoft Windows Vista and Mac OS X included voice recognition and speech synthesis features deep in their operating systems. However, due to the great hardware resources required and the decreasing reliability of such systems when the number of terms grows,

both systems limit the functionality of the voice recognition [11]. Therefore, voice command/voice dictation have not being utilized by the majority of users.
Recently, the smartphone industry has tried to provide a better VUI. Thanks to new trends in cloud computing and information retrieval, speech recognition can now be handled at a remote data center using n-gram voice recognition, and the queries can be understood through keyword searches/ tagging approach, instead of matching a list of predefined commands [12]. This allows speech recognition system to leverage a wide variety of vocabularies, tolerate error inputs the voice recognition component, support a variety of sentence structure, and utilize the computing power of the cloud to improve speech recognition algorithms based on the user's pronunciation [13]. However, it is self-evident that with a larger vocabulary, voice recognition has a higher error rate than it would with a smaller vocabulary.
On the other hand, the use of cloud computing implies a limited user interface which can only be operated under an Internet connection. It also provides a poor experience when the Internet connection has high latency [14]. This is problematic, as wearable devices have a limited supply of power [15], which restricts the devices to provide a high speed, low latency Internet access for the VUI, while maintaining a satisfactory battery life.
Another challenge of using n-gram voice recognition and keyword tagging strategy is the problem of "misheard". All through human history, misheard has become a part of our culture. Most notably example including Taylor Swift's "Blank Space", where "got a long list of ex-lover" being misheard by millions as "got all lonely Starbucks lover", and the fake game show "Homonym" in TV series 30 Rock, where the player has to guess a word from the sound, which is impossible as the game chosen homonym, so for one pronunciation, there are multiple possible meaning.
While in theory, n-gram voice recognition can resolved this problem by looking for frequent words pattern. However, in reality the correctness of the result is determined by keyword tagging approach, which leads the n-gram model in the voice recognition system to be trained with huge amount errors, which then lead it to make many incorrect recognition errors as shown in the the figure.
This final year project aims to develop a voice command-orientated interface which is optimised for power consumption and other limitations posed by wearable devices.


Taylor Swift
@taylorswift13
Follow

Sending my love to all the lonely Starbucks lovers out there this Valentine's Day.....even though that is not the correct lyric.

7:50 AM - 15 Feb 2015

113,279 RETWEETS 189,874 FAVORITES

Figure 1: Taylor Swift referring the misheard of her lyric on Twitter

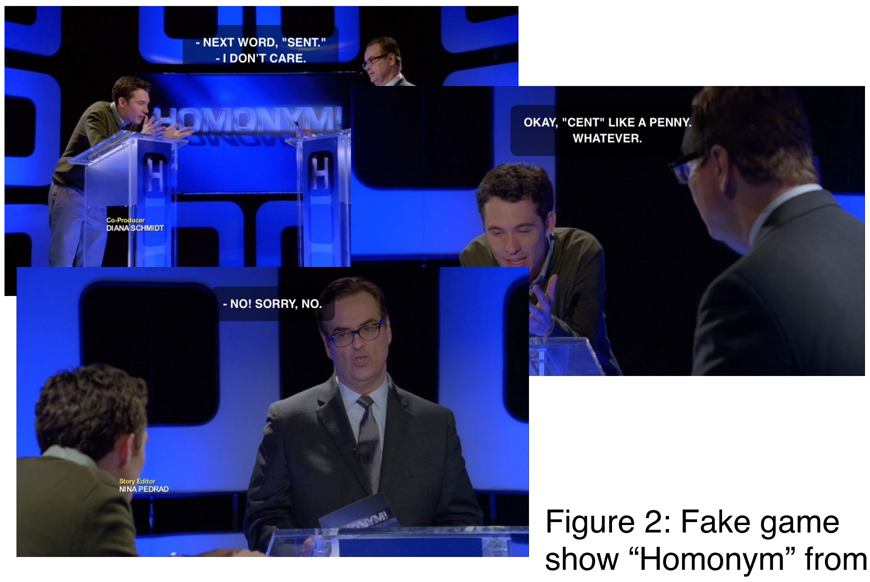


Figure 2: Fake game show "Homonym" from TV show "30 Rock"

1.2 Objectives

The goal of this project is to construct a user interface for a new type of wearable device, which is designed to optimize voice input. Because of extensive research in academia and in industry, the structures of speech recognition and speech synthesis algorithms are already highly developed. Therefore, these aspects will not be the focus of this project. Instead, we are focusing on how to parse the user's voice input, how to trace the object reference in the input, and how to handle notifications. To achieve these tasks, the project includes the following features/changes in the VUI:

- Remove all components that require graphic output.
- Notify users when notifications are received/triggered.
- Allow extensions to be attached to the VUI, which provide vocabulary, objects and services that make the VUI more extensible.
- Allow on the fly extensions: allow extensions to load and upload while the interface is operating.
- Generate statement hypotheses based on terms that possibly appeared and then reject grammatically incorrect or illogical ones.
- Evaluate user statements based on conversation context (**"he"**, **"she"**, **"it"**, **"here"**, **"the cinema"**, **"Who got the highest score?"** etc.)

If time permits, the following features will also be included:

- Allow queries to interact between objects from multiple extension
- Delay Notification: allowing the extensions to process the user requests in the background and report the results when result is available.
- Multi-layer interface, i.e. allow the interface queries to be directed to a speech recognition server for unfamiliar term
- Conditional Loading: this feature allows services in extensions to load, unload or install based on predetermined conditions (location, time, nearby person, certain phase appear in query)

1.3 Literature Survey

1.3.1 Currently available Wearable Device Interfaces

1. Virtual Reality with a GUI - Google Glass

   Google Glass is a wearable device developed by Google X, a division of Google, which focuses on technological advancement. It includes a display mounted on the user's head, which displays a GUI similar to the one in figure 1. It also features a camera, and it allows the user to interact with a touchpad or use voice input [16]. While Google Glass has received several acclaimed reports from media outlets, many reporters complain that the device consume too much battery power, as the device can only be used for around five to six hours without recharge [17].

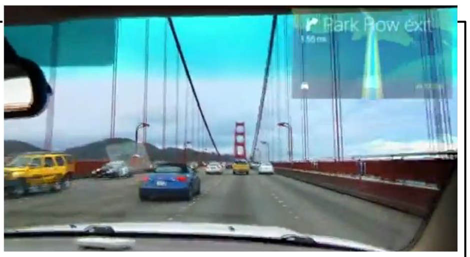


Figure 3, Google Glass user interface

2. Traditional GUI - Moto 360

   Moto 360 is a smart watch released by Motorola, developed while Motorola was a subsidy of Google. It includes a circular touch-enabled display with two different sizes. It is able to display the interface of Android Wear, a version of the Android Operating System designed for wearable devices. Android Wear includes display notifications [18], and accepts voice searches through Google Now. It requires a GUI similar to figure 2 [19]. However, similar to Google Glass, Moto 360 also features limited battery power [20].

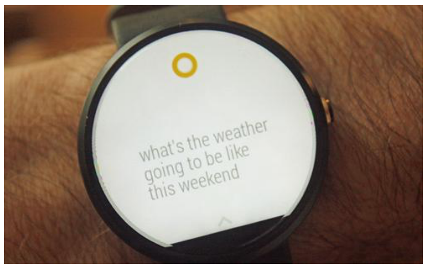


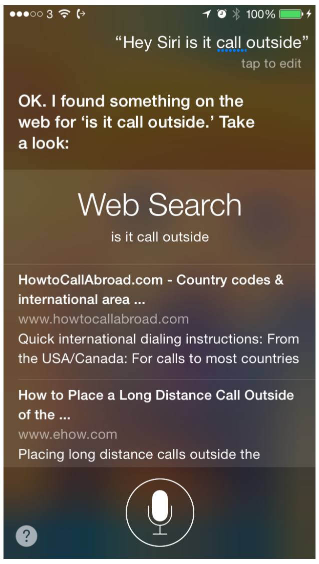
Figure 4: Moto 360 user interface for Google Now

1.3.2 Natural Language Processing Approach

1. Keyword Tagging Approach - Apple Siri

“People can say the same thing in a number of different ways, making sentence interpretation a Herculean effort, so Siri looked for keywords and their context instead. This simpler paradigm, along with a host of programmed phrases and requests it was designed to recognize and carry out, allowed Siri to guess what the user was asking and respond appropriately without having to understand every single word -- with a fair amount of accuracy.” - Bernadette Johnson, on how Siri works [21]

Apple Siri is virtual assistant software first developed by SRI International [22]. In April 2010, its subsidy, Siri Inc., was acquired by Apple Inc., which later released Siri as a preloaded application with iPhone 4s and iPad 3rd Generation [23].



“Hey Siri is it call outside”
tap to edit

OK. I found something on the web for ‘is it call outside.’ Take a look:

Web Search
is it call outside

HowtoCallAbroad.com - Country codes & international area ...
www.howtocallabroad.com
Quick international dialing instructions: From the USA/Canada: For calls to most countries

How to Place a Long Distance Call Outside of the ...
www.ehow.com
Placing long distance calls outside the

Figure 5: The input was “**HEY SIRI IS IT COLD OUTSIDE**”, which should display the information about the weather, not search.

The working principle of Siri can be separated into two stages: preparation and interaction. During the preparation stage, Siri requests device-specific identification from a backend server. Then Siri uploads the name of the contact, installed applications and songs in the device as a part of the user-define vocabulary [24]. When a user interact with Siri, it will record the user speech, compress and transmit the speech to a server responsible for speech recognition service, which has a large vocabulary list, alongside the device-specific identification to link the speech to the user-define vocabulary and parameter on user pronoun action generated by past requests [24][25]. The servers then respond to Siri with a list of possible statements, from which Siri can search each for keyword until statement contain specific action, which will be executed by Siri with graphic and voice response [24].

While Siri has been developed over ten years with US Military funding, the critics’ response is mildly negative. Report cites its poor voice recognition results [26]. As Siri will determine its action based on keywords it could find. If the keywords are not being recognized, the query could not perform as shown by figure 3 [21].

Along with other limitations Siri also a lack of extendable ability. While there are 1.5 millions applications available for iOS device, none of them can provide service, vocabularies, or data to Siri, which limits the feature available. Another reason behind this is the use of the keywords approach as when the developers add keyword which already exist in the system or other applications, they cause a conflicting situation between which application to use. It also creates a security vulnerability where an application could list all the keyword and prevent the system being functional.

2. Search Engine Backend Approach - Google Now

As mention in Moto 360, Google Now is an application in Google Android, which allow user perform voice search in natural language. When a user communicate with the application, the application would directly pass the audio record to a Google Search server, which first translate the speech into text, then pass directly into Google Search, as Google Search has the ability to isolate the keywords from elements of language, display the result in a card form interface, and read aloud the answer in some cases. However, while Google Now has the ability to perform voice based interactive, Google Now is focused on providing graphic based suggestion to users, which highly depended on the graphical user interface. On the other hand, Google Now's ability of natural language processing is based on isolating words from search query, instead of understanding the questions or statements from user. Therefore, we could not use Google Now on query similar to **"CALL MY SISTER'S STUDENT"**, and limited the functionality as a user interface. Finally, Google Now solely depended on Google Search, which makes this approach infeasible on wearable device when there is minimal or no Internet connection.

3. Voice command approach - Apple Speakable Item

   Apple Speakable Item is a voice command installed in Mac OS X 10.0 - Mac OS X 10.6, based on Apple PlainTalk, a voice recognition and synthesis framework, which developed to showcase, the ability of DSP installed in the Macintosh. Apple Speakable Item defined a set of sentences that could be detected, and the action should be performed. It also provides an API for third party applications provide voice commands.
   Because of its rigid command definition, Apple Speakable Item could achieve a reasonable accuracy with minimum hardware resource, without users performing a training process. However, the downside is users have to memorize all of the commands. The users could not use command with any changes. (**"OPEN THAT FILE"** is not equivalent to **"OPEN FILE"**.) This limit the objects users could control in the interface.

4. Hybrid Mode Approach (Keyword Tagging and Voice Command) - Microsoft Cortana

   Microsoft Cortana is an intelligent personal assistance application developed by Microsoft for Microsoft Windows Phone 8.1. Compared to Siri and Google Now, it features the ability for third party application developers to develop application that could provide service in the voice-based user interface. Also, Microsoft Cortana is famous for its ability to handle query referencing to previous query. For example, Microsoft

Cortana understands what is referring by the words **“HE”, “SHE”** and **“IT”**.
However, the implementation for system service and third party application are implementation in different fashion. In order to provide a service in Microsoft Cortana, the developer has to define the grammar and vocabulary of the voice command, which is a similar approach to the traditional voice-command user interface [28]. This prevent the interface provide interact with different application. On the other hand, due to different implementation by different applications, users have to learn different grammar and vocabulary. Therefore, the interface could not provide a single style of grammar and vocabulary.

5. Big Dictionary approach - IBM Watson

IBM Watson is an artificially intelligent computer system, developed by IBM as a part of its DeepQA project. The system is capable of processing questions in natural language by analyzing and enumerating all the possible word phase each word in the question as deep parsing, and generate answers based on 4 terabytes of data from different sources, including books and Wikipedia. However, while the system is highly optimized, the system has used 2,880 POWER7 processor cores and has 16 terabytes of RAM [29] , which is infeasible to install in wearable device, or any a desktop computer. It is also unsuitable as a VUI, as it only accept question as input, while a VUI for wearable devices a wider variety of sentence structure, including commands and exclamations.

2. Design

The Design Phase of the project started in early June.
Based on the research, we have developed the architecture of the project (Appx. B), and it is being executed in the following manner.

2.1 Design vocabulary token

The traditional approach to the parsing phrases is a static method, i.e. all vocabulary used in the system would be determined in the design phrase, and leave no room for adding/removing vocabulary dynamically during runtime. However, in this project, the dictionary will continuously grow and shrink as the user and system continues to activate and deactivate third party extensions, VUI require a grammar analysis engine (Grammar Engine) that based on a token that contains the information of each phrase (the extension that introduce the phrase, the id of the phrase, and the part of speech of the phrase).
The vocabulary have been grouped into two sets: predefined constants, and dynamic vocabularies originating from third party extensions.
Predefined constants include simple pronouns such as we, I, he, she, etc. It also includes variations of "be" (is, am, are, etc.), question words (what, how, who, where, etc.), and prepositions (of, for, etc.). These phrases are chosen as constant because of their fundamental nature in English and also to allow the main user interface to resolve pronouns fast. Therefore, they are pass directly to the grammar engine, and back to the user interface.
Dynamic vocabularies are added at runtime through either third party extensions, or when the system detects new vocabularies being used in the user interface. Every dynamic vocabulary is associated with a token that includes the identification number of the extension or the object type, the vocabulary type (verb, noun, adjective, etc.), and a unique identification number that associates the meaning of the word.

2.2 Design architecture for implementing "meaning" in a dynamic structure

In this project, every actions, description, and objects are defined in a type of plain XML text, called the property list. When a word is referenced, the tag of the word will be parsed into an object of GCObjectRequest, or its subclasses. GCObject will then trace the definition of the request, in the form of a dictionary data type. The definition dictionary will then be passed to the target object of the request, along with object of type NSExpression or NSPredicate to either perform the function call or evaluate the value defined in the definition. The architecture is designed this way to ensure the definition can be shared and consumed across extensions, hence allow the reuse of the definition from one extension to another, which can also enable scenarios such as cross extension queries. For example, a lyric extension could use the dictionary to access the attributes of a song, and use it to generate the actual lyrics.

2.3 Design a method to convert synonyms into a set of predefined words

We have designed an architecture for synonyms to reference an implementation for its meaning. For example, when referring to a physical object, **“BUY”** and **“PURCHASE”** have equal meaning, and have been referred to the same method to handle.

2.4 Design a method to handle multi-word phases

There is many proper nouns constituted from multiple words. For example, “Taylor Swift”, “Hong Kong”, “The Big Bang Theory”. While in n-grams voice recognition such phases has a greater score, the voice recognition is still possible to retrieve only a partial of the phase, which should be rejected by the system because it is illogical.We should send the result into the grammar engine with merging the partial words into one single word, which makes the grammar engine compatible with these multi-word phases.
The intuition way to achieve this is restricting the voice recognition engine to handle multi-word phase as a single word. However, this could not achieved in the modern voice recognition engine, as the vocabulary mainly accept independent words, without a silent period in between.
Therefore, we have been implemented an approach, which define a multi-word grammar for the voice recognition, to validate the result based on the position of the word, relative to the length of the phrases. (Hence “Taylor Kong”, “Hong Swift” is considered validate result in this process, but “The Kong Bang Theory” is not.)
This provide the system an accurate way to handle multi-word phases as a single word, while without modify the voice recognition system.

2.5 Design a data structure for storing and accessing context

We have designed a data structure to store a reference to the items, the query and the result of the query to simulate the context in a human-human conversation. This allows the project handle inputs such as **“WHERE IS IT?”, “DID HE TOOK IT?”.** It also recognizes other type of input based on pronouns and other terms which implicitly referencing the items mentioned in previous input.

2.6 Design an algorithm to reject failed statements

Most voice recognition system or machine learning algorithms retain errors while processing input. This is normally countered by retrieving multiple results from the system, and rejecting results that are not logically correct ranked from the highest score of the result. In similar means, we have designed an algorithm rejecting statement by throwing and catching exception during statement process period.

2.7 Design extension architecture

As the objects from different extensions interact in natural language (**“Play Taylor Swift newest album. ”, where “newest” as a term from the implement of basic object, and the rest by the media player extension**), we will design a structure capable to perform such query.

In order to design such architecture, we reference the XPC Services API by Apple Inc.
Apple XPC Service API is an architecture created in the effort of provide security and separate fragile code from the main program. While the benefit of this API is not related to the project, it provides a template on interacting with remote objects by function call or attribute access. By implementing a similar structure in the project, it enables the user interface to interact with third party extensions, and third party extensions interact with each other. This also provide a starting point for cross extensions actions.

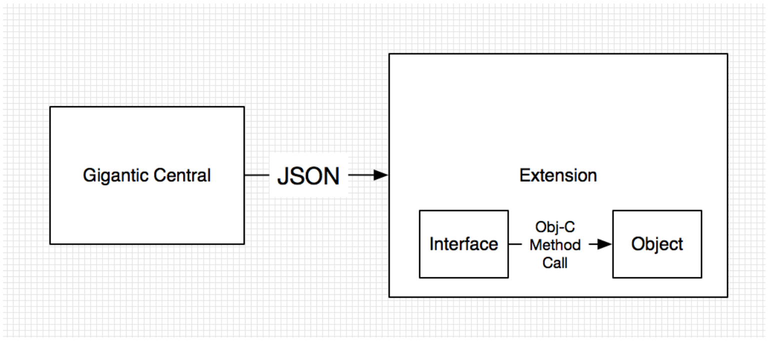


Figure 6: extension strucutre

2.8 Design project architecture

Traditionally, voice user interface are designed as a single process, where the command parser, the interpreter, and all the functions responsible for data are combine into one single process.
This approach not only limited the potential on upgrading, by requiring recompiling the whole process. It also limited the improvement third party developers could perform, as they need to reverse engineers the whole process, figure out which module they would like to improve, and inject their code into the process.
On the other hand, stableness is a key for any user interface design, as it is the gateway between the user and the device. Therefore, in this project, we would design a architecture to absorb any impact on crashes or error.
As the traditional structure of voice assistance resemble the structure of monolithic kernel, we decided to reference the other type of kernel structure: micro kernel, for the project architecture.
Micro kernel is a kernel structure, which reduce the functionality of the kernel to the bare minimum, like basic IPC, memory management, etc. It enables developer adding a new service does not require modifying the kernel, and typically results in a more reliable operating system, by a simpler design. [30]

Therefore, this project will follow the philosophy of micro kernel, and has the following design.

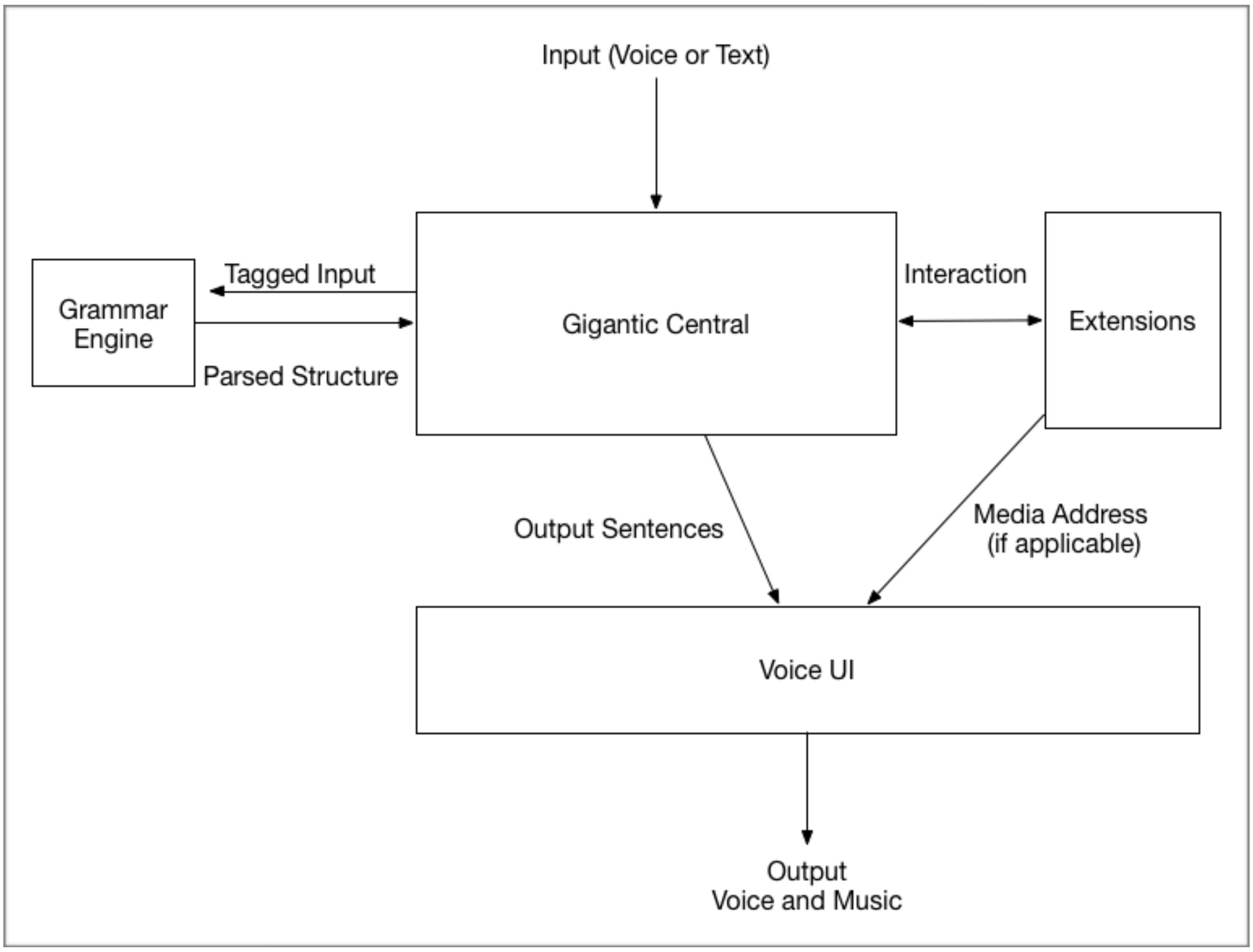


Figure 7: The process grouping of the project

The project will be separated into four different components:
The Gigantic Central will act as the kernel of the project, with the features of accepting the input, storing past referenced objects, schedule actions and value retrieval, and coordinating extensions to interact with the central and each other.
The Grammar Engine will act as the firewall of input, where any grammatically inappropriate input will immediately rejected.
The extensions will act as the “drivers” of the project, where any interaction with data, including data transformation, and perform actions of an object, could be defined by any third party developers.
Finally, the Voice UI aggregate any output into one coordinate output, similar to the design of console terminal. The Voice UI will schedule and coordinate outputs and music, to create a consistent result.

2.9 Design the communication channels between components

As shown in 2.8, in this project, the project into four separate processes, to ensure reduce the hardness of development. Therefore, we also need to define communication channels between components.

the grammar engine, and the core are connected via TCP connection. This ensure the core component is noticed if any error or crashes occur on the grammar engine, which is a vital part for the core to provide its features to the user.
In the project, every extensions are connected to the core (Gigantic Central) of the software, by using a client server model, via Unix connection. While we could reduce the cost of the communication, by switching to Unix domain socket, as they do not require the package being encapsulated with frame, and performing checksum verification, it lacks the ability for the

2.10 Design solution for the “Her” problem

As we all known, the word "her" has two meaning: as a ownership pronoun of something owned by a female, or when referencing a female as the "object" in a sentences (, as opposed to the "subject"). As this project is enforcing "strong grammar" (as any input should formed using a set of predesignated grammar), the grammar engine need to distinguish between each meaning. To achieve this goal, a new word "her1" as added in the grammar engine, and core (Gigantic Central), which responsible for the second meaning of "her", where the first meaning is solely represented by the original word "her". And when text input, or the voice input accepted "her" as a part of user query, each version of "her" will be created as a possible hypotheses.
In this design, as user could use one meaning in natural language, at most one of the hypothesis will be accepted by the grammar engine, and the other would be rejected as input which is inconsistent with the grammar rule. Then, the correct meaning will go through the core to generate output.

3. Implementation

The Implementation Phase included the following aspects:

3.1 Implement the phrase token and its algorithm

In the system, every phrase except the predefined set of keywords is respected by a vocabulary token. Phrase token are in the format of <extension which import the vocabulary>_<part of speech symbol of the vocabulary> <identification number of the vocabulary>.
Speech symbol are currently implemented as the following:

Part of speech	Part of speech symbol
Verb	v
Common Noun	n
Proper Noun	i
Adjective	a

In order to support multi-word phrases with words based voice recognition system, each word of the phrase will be represent by a separate token, in the format of <extension which import the vocabulary>_<part of speech symbol of the vocabulary> <identification number of the vocabulary>.<index of the word in the phrase>_<length of the phrase>. Then the results of the voice recognition is passed to the algorithm which uses the tokens to verify whether the words constitute as a phase.
For example, when the token of "taylor" and "swift" are "1000_i0.0_2" and "1000_i0.1_2" respectively, a "1000_i0.0_2 1000_i0.1_2" would be equivalent to "1000_i0", which is point the first proper noun in extension whose id is 1000. However, for a "1000_i0.0_2 1001_i0.3_4" input, the input would be rejected as the first part of the tokens are not identical, the index of the word is not in sequence, and the length of the word is not equal.
We also need an algorithm, to trace objects based on a string of phrase tokens, and the relationship symbols ("->" represent the relationship of relationship, "&&" represent "and", "||" represent "or"). For example, when input a string of "1000_i0->1000_n1" into the algorithm, where "1000_i0" represent an instance of a singer, and "1000_n1" represent the common noun "album", the system should return a set containing all the albums of such singer.

3.2 Implement the Grammar Engine

The mainstream voice assistance is lack of a structure of command and question, which increase the difficulty and complexity on implement third party voice commands. It also prevent the voice command along multiple applications. Therefore, we have developed a Grammar Engine, which is responsible to parse the input into a formatted structure as the following:
Input: 1000_v0 1000_i0 123_a5 1000_n1

Output:

```
Type: Command
Action: 1000, 4, 0
Target: 1000, 10, 0->1000, 1, 1|123, 3, 5
```

Input: What 0_n0 is it

Output:

```
Type: Question
Question Type: 0, 1, 0
Target: it->0, 1, 0
```

The output contains the following structure

Key of the message	Value of the message	Usage
Type	Command Question Statement Exclamation	[Type] represent the sentence type of the input.
Target	A string of phrase tokens, and the relationship symbols	[Target] will used to trace the actual objects/attribute, and used to apply constraint on it
Action	A phrase token represent the action	[Action] is used to get the dictionary describe the action
Question Type	A phrase token of a proper noun	[Question Type] is used to remove query result that does not fit the question. e.g. a phrase token of people will remove all objects in the result that is not a person.

Based on our design, flex and bison was used to write an algorithm responsible for parsing the sentences into a proper data structure. Flex and Bison has been chosen because of its ability to build compiler, in which they have demonstrated their power to build a light weighted algorithm based on a set of predefine grammar in computer language. Beside parsing sentences into a formatted text, Grammar Engine also includes code which would reject the sentences in an improper sentences structure (for example, “Album play new” has the linguistic structure of <object><verb><adjective>, which is not a valid sentences structure), and response a “Fail” message.

As decided in the design phase, the Grammar Engine is implemented as a separate, stateless component, which communicated with Gigantic Central. In the implementation, they are communicated via a TCP socket. This enables the Central keep track of the condition of the Grammar Engine, and allows Grammar Engine being hot swap dynamically at runtime. This is beneficial as this allows the Grammar Engine to be updated, restarted, or change to another language without reseting the Grammar Central, which act as the hub of the project. As a result, it enables the project change to target another language, or dialects, with the minimum changes in the project, in the future.

3.3 Develop the central component of the project

A program called "Gigantic Central" forms the workspace for the objects to be retrieved, rearranged and execute actions.

The elements in the interface need to interact with each other. This needs were fulfilled very neatly through Objective-C and Objective-C++, a language designed to support dynamic dispatch: the process for a program to select the implementation of a method when it is running. It also support method forwarding This is a crucial part of the project as it reduced the complexity to implement services, objects, features and functions without limiting the vocabulary users could use to represent the same meaning.

Gigantic Central provides three features: input processing, query process and extension interaction.

i. Input Processing

Gigantic Central contains two input method: voice input by trigger a voice recognition library to start voice recognition, and normal text input, which is used for test and evaluation. In the process, Gigantic Central will used the algorithm implemented in 3.1. The algorithm returns a list of possible hypothesis, which is each send to Grammar Engine one by one, until a hypothesis can be successfully processed as a query.

ii. Query Process

After Grammar Engine has been processed a hypothesis, and the hypothesis is deemed grammatically correct, the Grammar Engine return a query request packet, which is parsed as an object (GCQuery). Then, Gigantic Central looks for the object referenced in the query. Afterward, the query will go through four stages:

A. Apply Constraints

The objects traced in the previous step is tested, if there are any constraints, and remove any objects that does not fulfill the constraints.

For example, when processing "oldest person", the query will only preserve the oldest person in the trace.

B. Apply Action

After filtering the result in the previous stage, every remaining object will be subjected to perform actions, which would return whether the action is successful or not. After every objects have been enumerated, Gigantic Central will then send a finalization notification to every class of the objects, and finalize the actions.

For example, when attempting to apply "play" to 5 songs, Gigantic Central will ask each songs to play, then Gigantic Central will send a finalization notification to the class of the song, and finalize the action.

After the finalization, the class could send back a customized result string, which the query will store and output as the respond to user input.

C. Analyze

If the objects have not been asked to perform actions, the query will analyze the result. This stage is used to evaluate the degree of adjective, which could come from query such as "how old am I", "how expensive is this".

Tge the result is then stored as the objects of the query, which will be used in the next stage.

D. Form Answer

In the stage of forming answer, Gigantic Central separates the strategy based on the type of query it is forming for.

For commands type of query, Gigantic Central looks for the whether all the actions has performed successfully, then form a output whether the actions are fully completed, partial completed, or completely failed.

For commands type of question, Gigantic Central will just output every objects remained after all three stage.

After outputting the result dialog, Gigantic Central will store the objects into a list of historical referenced objects, which allows Gigantic Central to find the objects pronouns is referencing.

iii. Extension interaction

There are two types of extension interaction occurring in the project: one for Gigantic Central with the extension, another for extensions communicate with each other

For the first type, the extension is connected with the Gigantic Central via a Unix datagram socket, with address /vui/socket/GCE

For the second type, the intuitive way is create a P2P connection method, which each extension is connected with each other in an one to one direction. However, this design is problematic, as the total number of extension will yeald the complexity of $O(n^2)$, which is costly as such connection is only used for comparison between different two extensions.

Therefore, for inter-extension connection, the connection will be achieved by relaying via Gigantic Central as following:

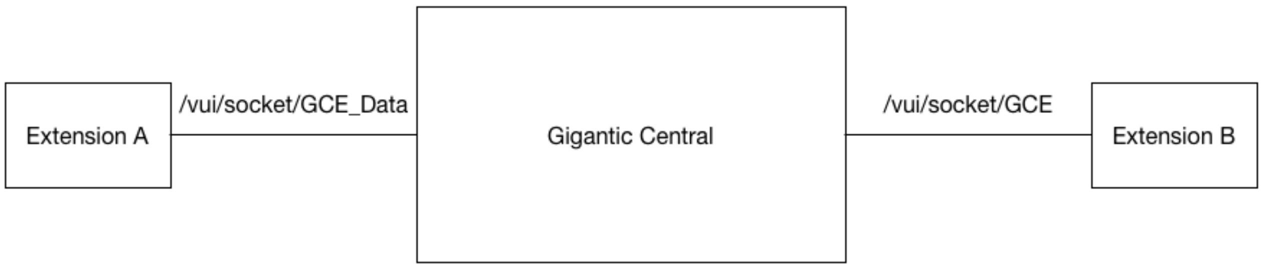


Figure 8: Extension A having inter-extension connection with Extension B, using the relay approach

In this architecture, the extension A connects with the Gigantic Central, using Unix datagram socket /vui/socket/GCE_Data, where any request by Extension A being relay to Extension B, and the reply being transmitted from Extension B to Extension A, by relaying by Gigantic Central.

3.4 Develop a protocol between Gigantic Central and third party extension

As the main objective of the project is to allow users to extend the interface with third party extension, we would need a connection protocol for the Gigantic Central access the objects in third party extension, perform method on such objects, and query the attribute of the objects.
In this project, Gigantic Central will open a Unix socket, which allows extensions to connect to the Gigantic Central, and exchange message in JSON message packages. When a connection started, Gigantic Central will first query the extension ID and the address of dictionary, then append the extension to a list. When Gigantic Central try to interact with the extension, the method call from Gigantic Central will be redirected, where the method call and the arguments are send to the extension, and await the extension to respond the result.

In order to keep the project with the least system dependency, the extension protocol does not use any available library. However, this would also enabled a tailored protocol, better suit for remote attribute process, and perform actions.
Request message:

Attribute Name	Attribute Type	Attribute Description
Command	String	The target command of the message
Request	String	The string representation of a tag, which is an object request
UUID	String	The string of UUID value of the object referenced in the extension
Objs	Array of string	The array contains multiple UUID value, which reference to objects in the extension
Action	String	The string representation of a tag, which represent an action
Session ID	Number	The number of a session ID, which provides
Key	String	The raw attribute name of the key e.g. age, name, numberOfTrack

Commands:

As there is multiple action the extension should perform, there is different type of command

Command	Description
Name	Return the extension name
Extension ID	Return the extension ID
Dictionary	Return the address of the extension vocabulary pronunciation, with the extension ".dict"
Instance Request	Return the instance requested by the central
Perform Action	Add the action and the object into a buffer, but **NOT** performing the action
Finalize	Finalize the action in the action buffer
Describe	Return the description value of the value (In Objective-C, description is a string that represents the contents of the receiving object.)
Sub object	Return the UUID of the sub objects (e.g. When requesting the sub object called "album", under the root object "taylor swift", the reply message will includes all the UUID of the taylor swift album)
attribute	Return the UUID of value for a key (e.g. When asking the age of the object, the command return the UUID of the attribute)
raw_attribute	Return the value for a key (e.g. When asking the age of the object, the command return the value of the attribute)
Route	Return the action/adjective "routing" in a dictionary
compareInHouse	The command will get the attribute using the UUID, then compare it directly, and reply the comparison result

Reply message:

Attribute Name	Attribute Type	Description
Result	String or Number	The basic result of the command
UUID	a string of UUID	The UUID of the return value
UUIDs	an array of UUID	Multiple object UUID from the command
Msg	String	The human-readable string after finalizing action

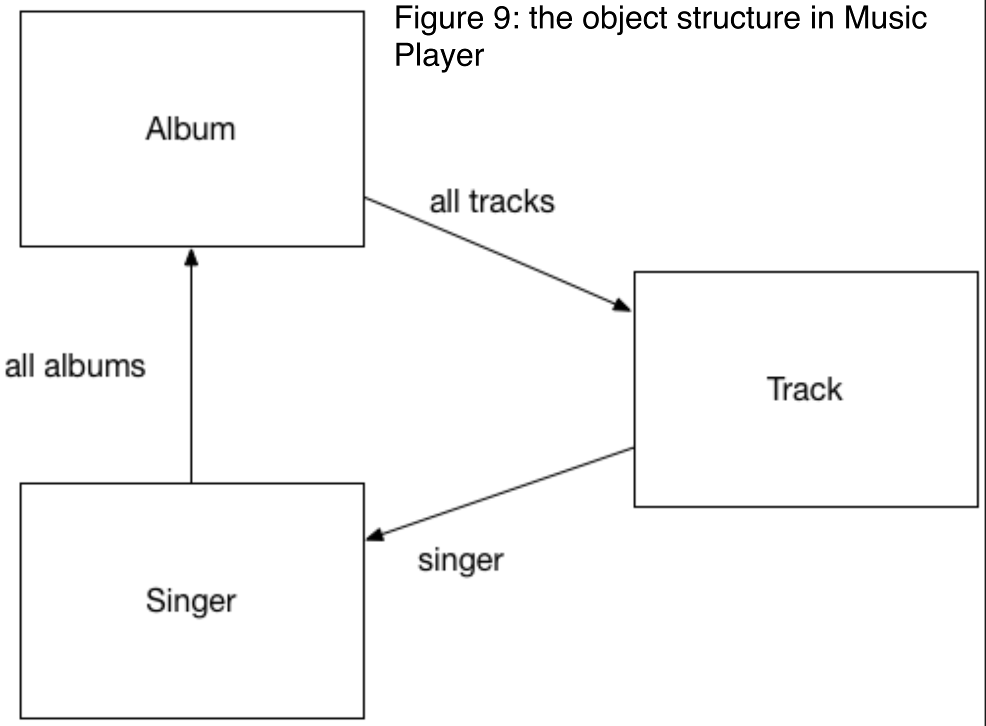
3.5 Implement sample extensions

In this project, we has implemented three extensions, to test, and show case the ability of cross-extension. The designed 3 extensions are music player, lyric finder, and restaurant finder.

i. Music Player

The “Music Player” extension define artists as “instances”, “album”, “single”, “music”, “song” as “nouns”, and “play” as “verb”. The object hierarchy is defined as following:

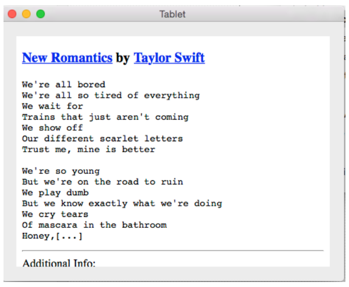
Figure 9: the object structure in Music Player



This is designed to test the ability of the project of tracing objects in extension, with demonstrate the ability of using adjective describe in the other part of the project.

ii. Lyric Finder

The “Lyric Finder” extension define only one noun: “lyric”. When referenced, the extension will look for the singer’s name and the title of the song from the “Music Player” extension. Then, the extension will ask a GUI application named “Tablet” to display the lyric. It is designed so it could stimulate the interaction between the wearable device and other digital devices.



New Romantics by Taylor Swift

```
We're all bored
We're all so tired of everything
We wait for
Trains that just aren't coming
We show off
Our different scarlet letters
Trust me, mine is better

We're so young
But we're on the road to ruin
We play dumb
But we know exactly what we're doing
We cry tears
Of mascara in the bathroom
Honey,[...]
```

Additional Info:

Figure 10: “Tablet” application, to demo as a remodel device

iii. Restaurant Finder

The “Restaurant Finder” extension one noun: “restaurant”, and a verb “find”. During the operation, the action “find” could be modified using “for <any person>”, which will filter any restaurant that has been blacklisted by any one in the modifier.

3.6 Implement an output program

In this project, there are three sources that output sounds to the user. These are the dialogue exchange, notification center and media player. As human ear can only listen a message at a time, a single output program can reduce the complexity dramatically. These benefits include limiting sources to "collide" with each other, provide priority on output, and fade in/fade out effect on the background music when an announcement/dialogue is initiated, to stimulate a natural transition for music to move from foreground to background.

The output program provides an Unix domain socket (/vui/socket/vui) for persistent connection from any application, which accepts a structured binary message. This message includes the notification or dialogue. When message is received, the program read the header in the structure to determine the priority of the message, then the message will be push into a list according to the priority.

The output program also provides another Unix domain socket (/vui/socket/musicPlayer), for media player to pass the address of the media files. When received at least one address, the program will save the addresses in an array, stop any playing file, then it will start playing from the first file, using AVMusicPlayer, a class in AVFoundation from OSX, to reduce the complexity of using third party music player. When all the file has been played, the program will stop the playback, and wait for the process to start again.

When there is a message in the list, the volume of the background music will be lowered. Then the message is send to speech synthesis system to output in speech. Once the speech is finish to output, the message will be removed from the list, and the volume of the background music resume.

3.7 Implement an voice output library

In 3.5, we have implemented an output program, which responsible for output the messages and music in a coherent way. Although it provides Unix domain sockets for direct socket communication, to facilitate the deployment of the code, we have also implemented an dynamic library, where the following functions have been implemented, using the sockets available.

```
void presentNotification(const char *summary, const char *description);
```

In this function, a notification message will be send to the output component, where the system synthesis a speech using the summary and description.

```
void presentDialog(const char *dialog, bool needFeedback);
```

In this function, a dialog message will be send to the output component, where the system synthesis the dialog, as an output of Gigantic Central.

```
void setPlaylist(const char *playlistPackage, unsigned int len);
```

In this function, the playlist array has been encapsulated with JSON, then it will send to the output component via /vui/socket/musicPlayer, to replace the current playlist.
If the playlist package is an empty JSON array, it will stop the music from playing.

Then, the library was used in the “Gigantic Central” for outputting the result, and “Music Player” for playing music.

3.8 Train acoustic model

As part of the project is using the logic of the system to filter out grammatically incorrect guesses from the voice recognition. It simplifies the difficulty to generate voice recognition guesses, by inputting a length of voice.
Therefore, we have used HTK Speech Recognition Toolkit to generate acoustic model, using the recording of 200 train sentences.

3.9 Asynchronous attribute fetching

In the current Internet economy, many attribute and features could be hosted online, especially in remote data center under cloud computing. Therefore, we need a model to fetch these attribute, which allows the VUI response to the request with a message which suggest the user the fetching is occurring. Once the attribute is ready, VUI will output the result as a voice notification.

4. Testing

During the development process, unit testings were written and executed periodically to ensure all modules were correctly working after every code commit. System integration testing is also being used to test all the components together as a project project.

Given the lack of GUI, and the nature of voice, it's relatively difficult to test the performance and correctness of the project using the available user interface testing tools. However, as this project is mainly focused on the logic of voice assistance, we would use text input, instead of voice input to perform the testing to reduce the variance on its performance.

4.1 Test the Grammar Engine

To test the grammar engine, we have prepared a list of correct sentences, which the grammar engine would understand in the following manner:

- **HOW ARE YOU**
- **WHAT 123_N1 IS IT**
- **GET ME THE 123_N1**
- **123_V1 123_A1 123_N1**
- **WHAT 0_N0 IS IT**

Then we have generated a list of incorrect sentences, by deriving from the correct sentences, by changing one to two tags.

Finally, we have created unit test case, which would compare the output of the Grammar Engine with the expected output.

4.2 Test the Gigantic Central system

The Gigantic Central contains multiple components, which are tested individually before being testing as a whole.

In the test period, we focus on five components: the object tracing algorithm, the pronoun tracing algorithm, value evaluation algorithm, action perform algorithm, and the extension connection implementation.

The first three algorithms are tested using a set of sample input strings, which use the name of the traced object to verify rightness. The sample string could be “1000_i1->1000_n1”, "1000_i1->1000_n1|1000_a1", “her->1000_n1”, etc. As the samples are tested with a constant set of object instance, the correctness can be tested via unit testing.

The action perform algorithm can be tested via unit testing by verifying predefined action tags and object tags.

The extension connection implementation is tested using a different approach by generating two separate programs that will run as client and server. The client and server are suppose to crash if the connection fails. To test the program, we execute the two programs and observe if any crash occur.

4.3 Test the speech response system

The speech response system requires extensive testing to ensure its correctness as it is responsible for reporting notification to the users, report dialogue and media playback.

To test this part of the system, we have built a small command program which generates numerous test notifications and dialogues. Then, we verified the result directly by piping the numerous test cases through the speech response system.
The correctness of media playback is ensured by Music Player Extension, as it is feature is music playback.

4.4 Test extensions

Extensions form a core part of the project, as it is the cornerstone of majority of features. However, given the difficulty to verify the correctness of the extensions themselves along with the extension connections at the Gigantic Central, they are tested manually by setting breakpoint at the crucial points of message the exchange. It is made possible as the messages themselves are encapsulated in JSON which is human readable. The messages are checked against the expected message, using the protocol definition described in part 3.4.
For independent features of the extensions, they are tested using debugger to call the functions. For example, the music playing feature of the Music Player Extension is tested by calling the playing function directly in the debugger. This approach is used because the features are designed to be light weighted, so it would be possible to test it manually.

4.5 Test the complete system

To test the system as an integrated system, we would separate into two approaches:
The first approach is using text input. As text input are significantly more reliable compare using the voice recognition, it will set a cornerstone of proofing the system rightness, by eliminating the possibility the voice recognition come back with no correct hypothesis.
The text input was translated into tag, by setting up a dictionary, and asking each classes to register its vocabulary. Then the text input will go through the query system as any voice recognition hypotheses.
On the other hand, we also need to test the system rejection when the query is improper. Therefore, we have generated a batch of example audio files from the speech synthesis system on Mac OS X Yosemite. Then, we pass the audio files into the Gigantic Central under voice recognition configuration, where it pass to the result to voice recognition system, grammar engine, and finally to the output system. By checking whether the system only responds to the correct hypotheses, we would check whether the project is correct.

5. Evaluation

After we have finished all the testing, we evaluated the system to check how well it fulfills our initial objectives.
While this project contains multiple functionality, voice command is the cornerstone of VUI which can be fairly evaluated in a scientific method.
We have chosen to compare the capability of handling different sentences structure between this project and Apple Siri, as it is considered as one of the state of the art in the voice recognition system.
As this project is an attempt to develop a new structure, only limited amount of vocabulary are imported into the system. Therefore, the following 12 testing sentences are generated based on the vocabulary, while the media library of the testing device will contain only multiple Taylor Swift albums, a contract entry called Jonny, and a entry called Sanki, which is noted as a Japanese female.

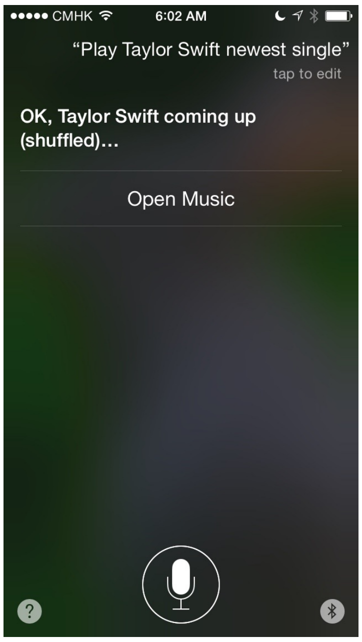
Music Player	**Location finding**	**Contact Analyze**
“Play Taylor Swift album”	“Find a restaurant”	“Who am I”
“Play Taylor Swift newest single”	"Find a Chinese restaurant”	“Am I old”
“What is the lyric of Taylor Swift new single”	"Find a Chinese restaurant for my Japanese friends”	"Am I Japanese”
1. “Play Taylor Swift newest single” 2. “What is its lyric”		“How old am I”
		1. “Who is Jonny” 2. “Call him”

In order to reduce the error caused by voice recognition in the evaluation process, we have manually input the sentences in the text form.
To compare the results from different platform, the following grading scheme will be used:

Score	Type of result
1	Command: Accurately execute the command without asking questions Question: The voice assistance return the correct answer
0.5	1. User interface try to confirm details of the input 2. VUI ask question before showing the result
0	The system ask user to manually analyze the input (e.g. in the third testing statement of the “location finding” section) Command: The command executed on wrong objects, or any fail existed Question: The system can’t retrieve the correct answer, or display a simple web search

The score has been accumulated, and used to evaluate the system against objective, and others similar VUIs.
Music Player:

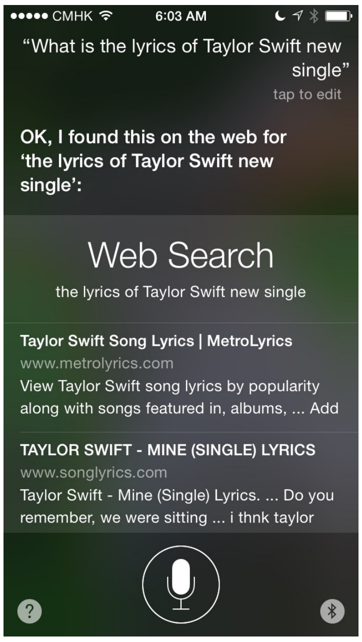
Query	Gigantic	Apple Siri
“Play Taylor Swift album”	1	1
“Play Taylor Swift newest single”	1	0 Reason: Wrong Action
“What is the lyric of Taylor Swift new single”	1	0 Reason: Result in a web search
1. “Play Taylor Swift newest single” **2. “What is its lyric”**	1	0 Reason: The first command can’t be executed
Total	4	1


●●●●● CMHK 6:02 AM

“Play Taylor Swift newest single”
tap to edit

OK, Taylor Swift coming up (shuffled)...

Open Music


●●●●● CMHK 6:03 AM

“What is the lyrics of Taylor Swift new single”
tap to edit

OK, I found this on the web for ‘the lyrics of Taylor Swift new single’:

Web Search
the lyrics of Taylor Swift new single

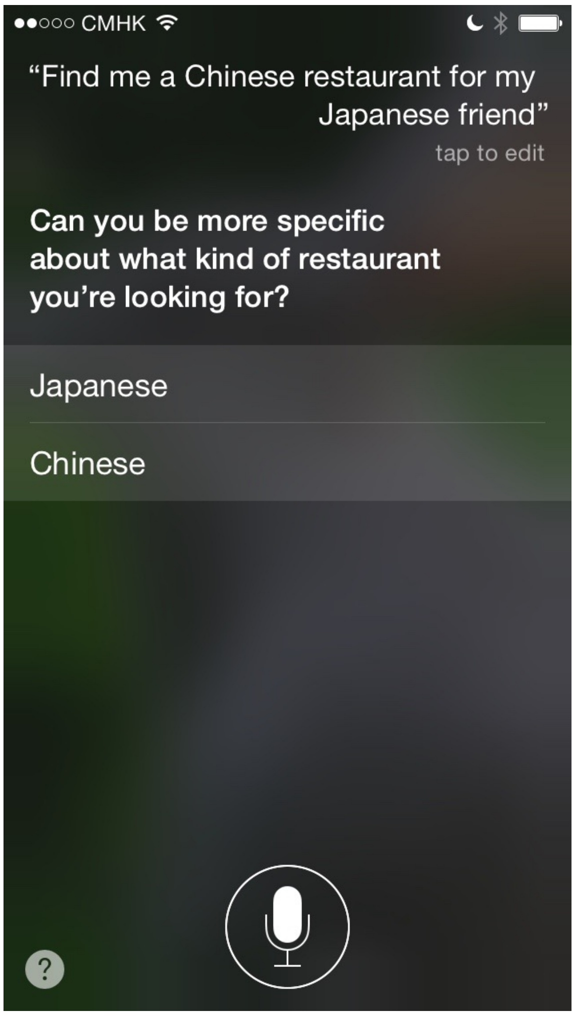
Taylor Swift Song Lyrics | MetroLyrics
www.metrolyrics.com
View Taylor Swift song lyrics by popularity along with songs featured in, albums, ... Add

TAYLOR SWIFT - MINE (SINGLE) LYRICS
www.songlyrics.com
Taylor Swift - Mine (Single) Lyrics. ... Do you remember, we were sitting ... i thnk taylor

Figure 10, 11: The sample return of Apple Siri under the query

Location finding:

Query	Gigantic	Apple Siri
“Find a restaurant”	1	1
"Find a Chinese restaurant”	1	1
"Find a Chinese restaurant for my Japanese friends”	1	0.5 Reason: Require confirmation on restaurant type
Total	3	2.5


●●○○○ CMHK

“Find me a Chinese restaurant for my Japanese friend”

tap to edit

Can you be more specific about what kind of restaurant you’re looking for?

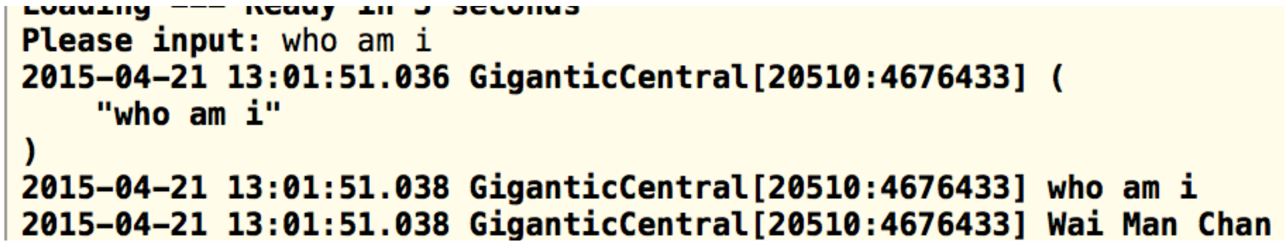
Japanese

Chinese

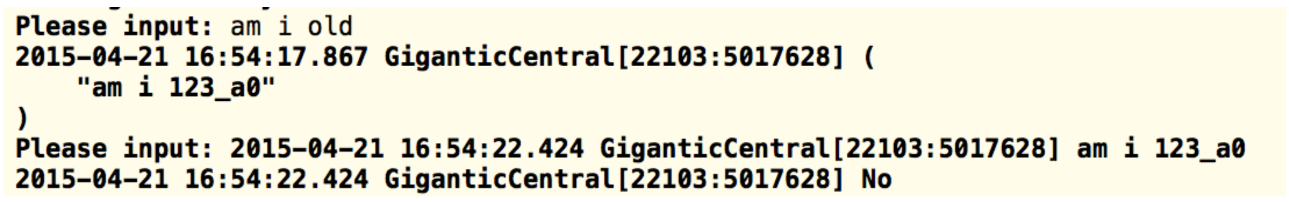
Figure 12: The sample return of Apple Siri under the query

Contact Analyze:

Query	Gigantic	Apple Siri
"Who am I"	1	1
"Am I old"	1	0 Reason: Return is not clear
"Am I Japanese"	1	0 Reason: Result in a web search
"How old am I"	1	1
"Who is Jonny" "Call him"	1	0 Reason: Siri ask for the caller
Total	5	2



```
Please input: who am i
2015-04-21 13:01:51.036 GiganticCentral[20510:4676433] (
    "who am i"
)
2015-04-21 13:01:51.038 GiganticCentral[20510:4676433] who am i
2015-04-21 13:01:51.038 GiganticCentral[20510:4676433] Wai Man Chan
```



```
Please input: am i old
2015-04-21 16:54:17.867 GiganticCentral[22103:5017628] (
    "am i 123_a0"
)
Please input: 2015-04-21 16:54:22.424 GiganticCentral[22103:5017628] am i 123_a0
2015-04-21 16:54:22.424 GiganticCentral[22103:5017628] No
```

Figure 13, 14: The sample return of Gigantic under the query

6. Discussion

6.1 Evaluation Result:

In the evaluation, it is clear the logic of the system implemented in this project has fulfilled the objectives. It is evidenced by Gigantic fulfill a satisfy requirement of all targeted testing sentences, as it get 12, while Apple Siri only achieved 5.5.

While the number of data source are significantly lower than Apple Siri. However, given the duration of a Final Year Project, and the manpower of an one-person team, compare to the resource of Apple, Gigantic posed a significant improvement against Siri, especially the pronoun tracing feature, and the ability of third party extension. It also simplify the development of VUI, as now extensions and developers could share their definition with each other, which reduce the number of redundant definitions.

6.2 Feasibility:

While some might suggest voice recognition is a performance demanding process, there is report where voice recognition system are feasible on mobile platform like PDAs and handheld device. On the other hand, many mobile phone, has equipped a voice recognition system for voice dial, before the commercialization of digital assistance, with a CPU with poorer energy efficiency. Therefore, it would be feasible to using a VUI offline.

On the other hand, the implementation of a offline VUI would allows the wearable device with a slow, but more energy conservative wireless and cellular connection, and remove the display and graphic processing unit in the wearable device. As a VUI could be designed to be activate with electronic signal of pressing a button, the cost of maintaining the VUI is low, compare to GUI, as the device need to constantly update the graphics in the GUI, and continuously display, until users manually set the device to sleep, or it reach a condition of auto sleeping.

6.3 The importance of having extension in VUI

While the majority of the VUI nowadays, provide a wide range of features and services, they are designed solely by companies who develop it, with their partner service providers. However, this type of development has limited the functionality of the VUI.

As of this moment, there is more than a millions of third party mobile application available in the market, and new services available everyday, it is obvious by enabling the extension ability, it will improve the VUI faster, than a single company.

6.4 Improvement achieved compared to command-based VUI approach

As the project is evolved from the command-based VUI approach, it resembles the command-based VUI. However, the following are two majority improvements upon the approach.

- Grammar is defined only once, in the Grammar Engine

The traditional command-based VUI require the developer define their own grammar, or the structure of the command. However, given we are using the grammar of natural language, implementing the grammar for every extension would be expensive, while meaningless.

- Vocabulary has be mixed between multiple extension

In the traditional approach, the vocabulary, and even the command, can not be shared between extensions. However, by making the functionality of extensions being used to import vocabulary, Gigantic has broken this limitation.

6.45 Improvement achieved compared to keyword-tagging VUI approach

Compare to the popular keyword tagging VUI approach, there are serval majority improvements upon the approach.

- Offline usable

Keyword tagging approach is currently used with a requirement of stable Internet connection, as it requires a high accuracy voice recognition engine, by using n-gram voice recognition with n=5, 6 However, by using offline voice recognition algorithm, we enables VUI to be network independent, as there are many locations where a stable Internet connection is not available. For example, lifts, underground car parks, majority of toilets in HKUST.

- Objects are traced, not tagged

In evaluation, it's clearly shown keyword tagging approach could be problematic when multiple contradicting keywords are available in a query. (e.g. "Find a **<u>Chinese</u>** restaurant for my <u>Japanese</u> friend)

In this project, we introduce a object tracing algorithm, which perform a Breadth-First Search, and trim the incorrect results after each level.

- Hypotheses contains homonym will now be properly rejected if it is logically incorrect

Simple command like "play music" and "is it cold outside" could be misheard into "pay music" and "is it call outside", because it is difficult to detect such error when the number of users is high, and the training data contains a proportion of error.

In Gigantic, such hypotheses will be first screened by the grammar engine, then test whether an action is proper, and whether the attribute exist. If any error exist, the query will reject the hypotheses automatically.

7. Conclusion

While this project is derived from the traditional command-based voice recognition, it has evolve from it, with the concept of third party development, cross-app functions, key value coding, and other modern ideas, which in combination, provided these features:

- object tracing under relationship
- pronoun tracing
- hot-swappable Grammar Engine
- Real time addition and removal of third party extension
- Voice Notification and dialogue based user interface

The majority accomplishments in this projects are the following:

- Object are traced based on their relationship, instead of the keyword tagging approach
- Object being filter based on adjectives
- Action could be modified, changing the effect and the result of the action
- Defining new features without defining grammar, and the
- Most importantly, the system could run without a remote system

However, there are many areas we could further studies, to solve the difficulty of implementation, and improve the user experience

1. Using big data to generate Grammar Engine

Currently, the Grammar Engine is generated using hand-written rule. However, with different languages with numerous of rule, we could simply generate the rule by data mining the association between type, by inputting a large amount of literature.

2. Tweaking voice recognition for the nature of Gigantic

Gigantic is sensitive to both grammatically and logically correctness. When linking with a n-gram voice recognition, it could generate multiple guesses which is grammatically incorrect. Therefore, it will be beneficial for the voice recognition to obey both n-gram and grammar rules at the hypotheses generating phase.

8. References

[1] JSBMarketResearch.com, “Global Smart Wearable Market Forecast and Opportunities, 2019.”, JSB Market Research; 29 Aug. 2014; http://www.jsbmarketresearch.com/technology/r-Global-Smart-Wearable-Market-Forecast-and-Opportunities-123160
[2] T Michael, K. Michael, A. Michael, “Wearable Technology and Wearable Devices: Everything You Need to Know,” Wearable Devices Magazine, 26 Mar. 2014; http://www.wearabledevices.com/what-is-a-wearable-device/
http://www.wearabledevices.com/what-is-a-wearable-device/
[3] Apple Inc, “Apple - Apple Watch,” Feb. 2015; https://www.apple.com/watch/
[4] Jawbone, “Jawbone Up,” Feb. 2015; https://jawbone.com/up
[5] D. Goldman, “Google unveils ‘Project Glass’ virtual-reality glasses”, CNN Money, 4 Apr. 2012;http://money.cnn.com/2012/04/04/technology/google-project-glass/?source=cnn_bin
[6] J. Reimer, “A History of GUI,” Ars Technica, 5 May 2005; http://arstechnica.com/features/2005/05/gui/
[7] NetApplications.com, “Desktop Operating System Market Share,” Net Market Share, Dec. 2014; http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0
[8] S. Pop, “Global AMOLED Market Will Expand Greatly in 2014,” Softpedia, 30 Dec. 2013; http://news.softpedia.com/news/Global-AMOLED-Market-Will-Expand-Greatly-in-2014-412897.shtml
[9] Google, “XE20.1 Release Notes” Google Glass, 20 Aug. 2014; https://support.google.com/glass/answer/6078902?hl=en
[10] J. Lewis, Practical Speech User Interface Design, CRC Press, 2010
[11] F. Qiao, J. Sherwin, R. Rosenfeld, “Small-Vocabulary Speech Recognition for Resource-Scarce Languages,” Proc. First ACM symposium on computing for development (ACM Dev 10), ACM, 2010; do:10.1145/1926180.1926184.
[12] R. Hof, “Meet The Guy Who Helped Google Beat Apple's Siri”, Forbes, 1 May 2013; http://www.forbes.com/sites/roberthof/2013/05/01/meet-the-guy-who-helped-google-beat-apples-siri/.
[13]D. Hardawar, “How Nuance tapped the cloud to bring voice recognition to mobile … and beyond,” Venture Beat, 8 Sep. 2013; http://venturebeat.com/2013/09/08/how-nuance-tapped-the-cloud-to-bring-voice-recognition-to-mobile/.
[14] P. Rubens, “Could Latency Kill the Cloud? ,” Enterprise Networking Planet, 22 Mar. 2012; http://www.enterprisenetworkingplanet.com/netsysm/could-latency-kill-the-cloud.html .
[15]L. Eadicicco, “Silicon Valley Never Talks About The Real Reason You Don't Own A Smart Watch Or 'Wearable Tech’,” Business Insider, 26 Mar. 2014; http://www.businessinsider.com/the-biggest-challenges-in-wearable-tech-2014-3 .
[16] Google, Google Glass, Jan. 2015; https://support.google.com/glass/?hl=en .
[17] Z. Epstein, “Google Glass is the future – and the future has awful battery life,” BGR, 1 May 2013; http://bgr.com/2013/05/01/google-glass-battery-life/.
[18] Motorola, “A watch for our times,” Jan. 2015; https://moto360.motorola.com .

[19] Google, “Android Wear” Feb. 2015; http://www.android.com/wear/.
[20] J. Stern, “Moto 360 Smartwatch Review: One Size Doesn't Fit All,” WSJ, 5 Sep. 2014; http://online.wsj.com/articles/moto-360-review-one-size-doesnt-fit-all-1409896802 .
[21] B. Johnson, “How Siri Works,” Feb. 2015; http://electronics.howstuffworks.com/gadgets/high-tech-gadgets/siri.htm .
[22] SRI International, “Siri, Inc. | SRI International” Feb. 2015; http://www.sri.com/engage/ventures/siri .
[23] Apple Inc, “About Siri,” Jan. 2015; http://support.apple.com/kb/HT4992?viewlocale=en_US&locale=en_US .
[24] Apple Inc, “Privacy - Privacy Build In,” Jan. 2015; http://www.apple.com/privacy/privacy-built-in/.
[25] S. Erick, “Siri's iPhone App Puts A Personal Assistant In Your Pocket,” TechCrunch, 04 Feb. 2010; http://techcrunch.com/2010/02/04/siri-iphone-personal-assistant/.
[26] L. Effron, “iPhone 4S's Siri Is Lost in Translation With Heavy Accents,” ABC News, 28 Oct. 2011; http://abcnews.go.com/Technology/siri-lost-translation-heavy-accents/story?id=14834111#.TwdUoZhSH8s .
[27] M. Kelly, “Were Apple’s Siri ads ‘false and misleading’?,” Washington Post, 16 Mar. 2012; http://www.washingtonpost.com/business/technology/were-apples-siri-ads-false-and-misleading/2012/03/13/gIQAtBBWGS_story.html?tid=pm_business_pop .
[28] Nokia, “Speech Enabled Calculator For Windows Phone 8 - Nokia Developer Wiki, “ Microsoft - Nokia Developer Wiki, 29 June 2013; http://developer.nokia.com/community/wiki/Speech_Enabled_Calculator_For_Windows_Phone_8 .
[29] Computer Science and Electrical Engineering Department of University of Maryland, Baltimore County, “Is Watson the smartest machine on earth? ,” 10 Feb. 2011; http://www.csee.umbc.edu/2011/02/is-watson-the-smartest-machine-on-earth/ .
[30] A. Silberschatz, “Operating system concepts”, 2013,Yale University, p. 81
[31] A. Lee and T. Kawahara, “Recent Development of Open-Source Speech Recognition Engine Julius ,” Nagoya Institute of Technology, 2009

9. Appendix A: Work Summary

June, 2014

- Two versions of Grammar Engine have been developed to be the proof-of-concept, on whether flex and bison could handle simple English grammar
- Search for a suitable voice recognition library - CMU PocketSphinx and Julius have been chosen
- Search for a language suitable for implement the project - Objective-C has been chosen

July, 2014

- Implement a demo version of extension system, allow the extension to attach and detach to the query system
- Search for a trained voice recognition data set
- Add the ability to accept object in extension, and verb from without prior defining in the Grammar Engine

August, 2014

- Start the implementation of extension root object type (GCObject), the object types that extension share to the query system in C. Including Location, Person, and the general type of object.
- Develop Voice Interface, which speak the sentences provided from VIKit, the API of Voice Interface

September, 2014

- Define the scope of the projects
- Design a new parsing result structure
- Construct a testing dictionary for voice recognition system

October, 2014

- Implement “action performing” functions in GCObject
- Implement “comparison” functions in GCObject

November, 2014

- Implement pronoun tracing functions
- Testing different acoustic model for analyze the speech synthesized audio file

December, 2014

- Revamp Grammar Engine for the changes in the system
- Implement object tracing functions
- Rewriting GCObject with Objective-C
- Implement extension functions callback

January, 2014

- Implement media player extension
- Adding background media playback functionality

February, 2014

- Design a new structure for notification and dialogue message
- Experiment with capability of handle greeting statement (“good morning”, “hello”, etc.)
- Writing progress report

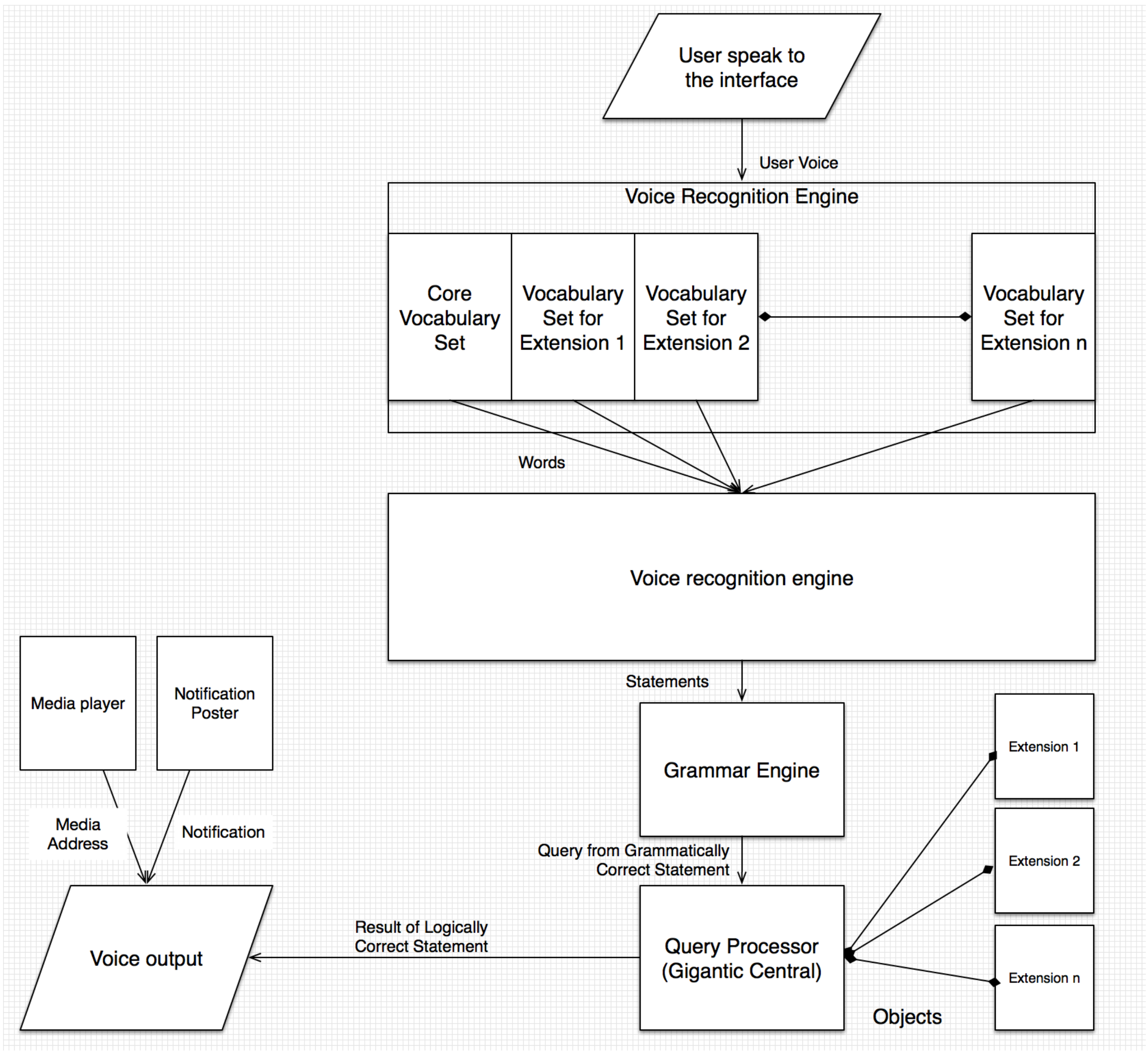
March, 2014

- Develop extension for music playing
- Implement inter-extension connections via P2P

April, 2014

- Develop extension for lyric searches and restaurant finder
- Re-implement inter-extension connection via relay
- Train and test acoustic model
- Writing final report

10. Appendix B: Project Structure


User speak to the interface
User Voice
Voice Recognition Engine
Core Vocabulary Set
Vocabulary Set for Extension 1
Vocabulary Set for Extension 2
Vocabulary Set for Extension n
Words
Voice recognition engine
Statements
Grammar Engine
Query from Grammatically Correct Statement
Query Processor (Gigantic Central)
Extension 1
Extension 2
Extension n
Objects
Result of Logically Correct Statement
Voice output
Media player
Notification Poster
Media Address
Notification

11. Appendix C: Sample Object Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
	<key>routingReference_verb</key>
	<array/>
	<key>routingReference_noun</key>
	<array>
		<string>age</string>
		<string>creationDate</string>
	</array>
	<key>routingReference_adjectie</key>
	<array>
		<string>old</string>
		<string>new</string>
	</array>
	<key>Verb</key>
	<array>
		<dict>
			<key>Command</key>
			<string>print</string>
			<key>Return Value</key>
			<false/>
		</dict>
	</array>
	<key>classID</key>
	<integer>123</integer>
	<key>Noun</key>
	<array>
		<dict>
			<key>Expression</key>
			<string>creationTime.durationToNow.year</string>
		</dict>
		<dict>
			<key>Expression</key>
			<string>creationTime</string>
		</dict>
	</array>
	<key>Adjective</key>
	<array>
		<dict>
			<key>asscending</key>
			<false/>
			<key>value</key>
			<string>age</string>
			<key>judgement</key>
			<string>ALL age.doubleValue &gt;= 60.0</string>
			<key>defaultNumber</key>
			<integer>1</integer>
		</dict>
		<dict>
			<key>asscending</key>
			<false/>
```

```
				<key>value</key>
				<string>creationDate</string>
				<key>defaultNumber</key>
				<integer>1</integer>
			</dict>
		</array>
</dict>
</plist>
```

12. Appendix D: Project Planning

12.1 Division of Work

Since this is a person team, there is no division of work.

12.2 GANTT Chart

Task	July	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr
Do the Literature Survey	■	■								
Analyze the possibility to use non-machine learning method	■	■								
Build the configuration file		■	■	■	■	■	■			
Choose Voice Recognition Method		■					■			
Design the Output of Voice Interface		■	■							
Develop Grammar Engine	■	■	■	■	■	■				
Develop the element of interface		■	■							
Design Extension API		■	■	■			■	■		
Develop sample Extension			■	■	■	■			■	■
Test the sample extension				■	■	■	■	■	■	■
Test the Voice Interface output		■	■	■	■	■	■	■	■	■
Test the grammar engine			■	■	■	■	■	■	■	■
Test the query system							■	■	■	■
Perform Integration Testing								■	■	■
Write the Proposal			■							
Write the Progress Report								■		
Write the Final Report										■
Prepare for the Presentation										■
Prepare an acoustic model for the Presentation										■
Design the Project Poster										■

13. Appendix D: Required Hardware & Software

13.1 Hardware

Development Machine	Macintosh with OS X Maverick or Yosemite
Installment Machine (Mac)	Macintosh with OS X Maverick or Yosemite, speakers and microphone are required

13.2 Software

OS X Maverick or Yosemite	For providing speech synthesis service on Macintosh
Xcode 6	Integrated Development Environment
Objective-C, Lex, Yacc, C++	Programming Language
Julius, HTK Speech Recognition Toolkit	For word based speech recognition algorithm
Flex, Bison, Clang	Compiler
GNUStep, Cocoa (Foundation API)	Implementation of Objective-C