Contents

[Justification 2](#_Toc513391920)

[1. Task domain vocabulary specification 2](#_Toc513391921)

[2. Interaction scenario specification / or defined from conducted observations. 2](#_Toc513391922)

[Both Scenarios 3](#_Toc513391923)

[Scenario One 3](#_Toc513391924)

[Scenario Two 3](#_Toc513391925)

[Network architecture 4](#_Toc513391926)

[3. Personality specification 4](#_Toc513391927)

[4. Grammar design 4](#_Toc513391928)

[5. Dialogue design 4](#_Toc513391929)

[6. Error catching / handling 5](#_Toc513391930)

[Error catching 5](#_Toc513391931)

[Error handling 5](#_Toc513391932)

[7. Action / response generation 5](#_Toc513391933)

[8. Evaluation with users / recommendations 6](#_Toc513391934)

[Description of work done 6](#_Toc513391935)

[Critical Evaluation 6](#_Toc513391936)

[Improvements 6](#_Toc513391937)

[Appendices 6](#_Toc513391938)

[Appendix 1 – Full system flowchart 6](#_Toc513391939)

3. Reporting. The 6p (max) report must follow standards of scientific reporting starting with a justification of the design reflecting background knowledge in the domain of spoken interfaces, then including a description of work done along the 8 points above, then a critical evaluation of the system’s performance. Use numbered sections. The DLE submission will be in the form of a zip file that also includes the software. Students can also submit videos on an online repository showing example of userevaluation (this is required in the case of a referred coursework).

# Justification

justification of the design reflecting background knowledge in the domain of spoken interfaces

Tree-based Graphical User Interface (GUI) frontends to database backends have been used for a very time (REFERENCE). For many medium-large databases, these interfaces can often prove to be frustrating (REFERENCE?).

Interaction with standard databases are only possible if the user knows how to execute SQL queries. GUIs that are constructed to allow untrained users to request information from a database generally fall in one of two categories: overwhelming or time-consuming.

Spoken interfaces can provide a faster, more intuitive method for accessing complex databases by untrained users(<http://www.rroij.com/open-access/database-interaction-using-automatic-speechrecognition.pdf>), whilst still providing methods of searching and exploring the database for users who don’t completely know what they are searching for yet.

This projects focus is the development of an interactive speech interface to a database of plants compiled by Plants For A Future (https://pfaf.org/user/Default.aspx), with the aim of producing a system in which untrained users can find a plant faster than they could with a traditional GUI interface.

## 1. Task domain vocabulary specification

/ e.g. from collected corpus.

How did you define the vocabulary in your grammar? Made it up? Observed people? Run tests to refine?

The vocabulary for this system was generated with an iterative approach. A couple of obvious sentences were constructed in the grammar. This grammar was then tested on people other than the author. Whenever an out-of-grammar sentence was spoken, it was written down, added to the grammar, and the iterative cycle was repeated on a new volunteer.

## 2. Interaction scenario specification / or defined from conducted observations.

This program is intended to serve as an assistant in garden centres. One or many kiosks would be setup around the garden centre, with either a built-in microphone and speaker or a tethered headset. This system has been developed to address two main scenarios.

### Both Scenarios

In both scenarios a user will either discover the kiosk or be shown to it by a member of staff at a centre. System voice outputs are highlighted in blue and user responses are highlighted in grey.

“Welcome to the talking plant finder. You are currently searching for plants that will grow in Britain. Would you like to change that?”

If the user says yes they will be asked which country they would like to change to, if not the program will continue.

"Would you like to find a plant to match an environment (normal search), or see what environments match a plant (reverse search)?"

Normal search will be outlined in scenario one, and reverse search in scenario two.

### Scenario One

In this scenario, a customer has an idea of where they are planning on growing a plant, but hasn’t yet found a plant that will grow well in these conditions.

“How can I help?”

The user can begin entering search criteria, or can ask for help.

“Do you have any suggestions?”

“Why not try searching for where the plant will grow, such as **East Wall**, or **Cultivated Beds**. Or you could try searching by type of plant, like **Bulb**, **Bamboo** or **Biennial**. You can also specify how hardy you want the plant to be.”

(Words in **bold** are randomly generated each time the user asks for suggestions)

“Okay, can you help me find an extremely hardy tree to grow against a shady edge?”

(Blue writing extracted from speech, SQL query automatically constructed:

**SELECT** `Common name`**,**`Latin name` **FROM** `plantlocations` **WHERE** `Habit`**=**'Tree' **AND** `ShadyEdge`**=**1 **AND** `Hardyness` **BETWEEN** 9 **AND** 10

Error checking and other validation performed here. After error checking:

“Okay, here are your results. Common name and then Latin name:

Puahou - Pseudopanax arboreus “

“Would you like more information on this plant?”

If the user says yes, they are directed to the reverse search.

### Scenario Two

In this scenario the user has a plant in mind but wishes to find out more about its ideal growing conditions. By knowing the common or Latin name of a plant, the user can perform a ‘reverse search’.

“Reverse search.”

"Please say the common or Latin name of a plant you want to search for."

“Garlic”

Generated query:

**SELECT** **\*** **FROM** `plantlocations` **WHERE** `Common name`**=**'Garlic' **GROUP** **BY** `Latin name`

Vocalised response:

“Latin name Allium sativum”

“Habitat Not known in a truly wild situation.”

“Habit Bulb” “Height 0.6” “Width 0.15” “Hardyness 8”

The user is then asked if they wish to use the service again. If yes, they are returned to a state where they can select ‘normal’ or ‘reverse search’ again.

Two main scenarios:

1. Person is wanting to find a plant that well grow well in their house/garden
2. Person has a plant in mind and wishes to know it’s ideal growing conditions

Person needs a plant, or person wants

Person is directed to machine

### Network architecture

Network architecture would depend on the size of the garden centre. For small centres, the program and database would be hosted on a relatively lightweight computing platform (such as a NetTop). For medium-sized garden centres, the database would be hosted on a central SQL server and NetTops would be distributed throughout the centre, each processing its own speech recognition and text-to-speech. For larger garden centres, a centralised SQL server and centralised nuance server would be setup, and users would interact with the system via thin clients placed around the garden centre.

To increase reliability, headsets would be the primary user interface. To further develop the system, microphone arrays could be used to cancel background noise and isolate an individual speaker(REFEERENCE - https://www.nttdocomo.co.jp/english/binary/pdf/corporate/technology/rd/technical\_journal/bn/vol9\_4/vol9\_4\_031en.pdf), removing the need for a headset.

## 3. Personality specification

Not overly formal but still proper

## 4. Grammar design

## 5. Dialogue design

(will add to the grammar)

## 6. Error catching / handling

(will add to grammar and dialogue)

The primary error catching is performed by the “envCheck1” state, and error handling is performed between the “envCheck” and “envErrorFix” states.

### Error catching

Error catching is primarily performed in the “envCheck1” state. This state accesses the arrays which have search terms stored in them and vocalises these search terms to the user. The user is then asked whether the spoken terms are correct. If the terms are correct, the number of matching search results are queried and vocalised. If there are any mistakes, the “envPreFixCheck” state is called which asks the user if a spoken term was missed (false negative detected), or if a term was recognised that the user did not say (false positive detected). For false positives, “envCheck” is called. For false negatives, the user is returned to the “freeSpeech” state.

### Error handling

“envCheck” starts by asking the user which of the recently vocalised terms was misheard. The user then says which terms were incorrectly identified and the system will remove them from its arrays.

To allow the user to speak multiple corrections in one go, grammar slots were created for each possible search term which are set to a ‘1’ when they get filled. An individual if statement was written for each of these slots. Below is an example of one such if statement:

NLGetIntSlotValue(AppGetNLResult(app), "perennial\_climber\_said", &perennial\_climber);

if (perennial\_climber == 1) {

printf("%s\n", "You said Perennial Climber.");

strcpy(errorStoreArray[errorEnvStringCounter], "Perennial Climber");

errorEnvStringCounter = errorEnvStringCounter + 1;

}

To detect when the user had finished speaking corrections, the following code was written:

if ((AppGetRecognitionStage(app) == 4)) {

AppGoto(app, "envErrorFix");

}

Where “ …) == 4)” corresponds to “AFTER\_SPEECH\_STAGE” in the enumerated “RecognitionStage” structure returned by “AppGetRecognitionStage”.

## 7. Action / response generation

This program relies heavily on the MySQL Connector/C (MySQL C API). This API provides a programming interface (in C) to communicate with an SQL server (hosted either locally or remotely). Data is requested by constructing SQL query strings and submitting them via a MySQL object created from the API. If the user had specified “Climber” and “East Wall” as search terms, the following SQL query could be generated:

**SELECT** **COUNT(**`Latin name`**)** **FROM** `plantlocations` **WHERE** `Climber`**=**1 **AND** `East Wall`**=**1

“Climber” and “East wall” are both column titles from the same `plantlocations` table, so this query is simple to construct in code. As the user selects categories from multiple tables the queries quickly get more complex.

To generate these queries, search terms spoken by a user are stored in various arrays depending on the category the search term belongs to. When the user requests their results (or a count of how many results match their currently selected search terms), the program will construct strings from these arrays in the format of SQL queries and submit them to the MySQL server containing the database. Results from the database are generally vocalised via nuances AppAppendTTSPrompt function.

## 8. Evaluation with users / recommendations

(also attach dialogue examples). ( see examples in project folder).

## Description of work done

* Installation and configuration of MySQL Connector/C (the MySQL C API)
* Visual Studio compiler and linker setup with nuance and MySQL Connector/C
* Configuration of Visual Studio remote process debugging
* Dialogue design
* State design
* Time and date generation and sentence construction
* String processing functions
* SQL query generation functions
* Initial design: Tree based approach
* Final design: free speech mode

## Critical Evaluation

of the systems performance

## Improvements

Check confidence levels. If confidence is below a threshold (say 70%), repeate all words that are above 55% back to the user and get them to chose the correct one (<https://dle.plymouth.ac.uk/pluginfile.php/1003364/mod_resource/content/0/ChoulartonDaleSST2004.pdf>)

Check prosidy:

<https://dle.plymouth.ac.uk/pluginfile.php/1003369/mod_resource/content/0/litman_2000.pdf>

# Appendices

## Appendix 1 – Full system flowchart

Imagine there’s a massive picture here…