Dynamic User Modelling For Speech Interfaces Using Implicit Personalisation

Steven Robert Denholm
H00017955

August 2014

Computer Science
School of Mathematical and Computer Sciences

Dissertation submitted as part of the requirements for the award of the degree of MSc in Software Engineering
Declaration

I, Steven Robert Denholm confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: ..................................................................

Date: .............................................................................
Acknowledgments

There are many people who helped during the course of this project. Firstly, I would like to thank my supervisor, Professor Oliver Lemon, for the advice and guidance given throughout the project, especially during the write-up of this thesis.

Secondly, I would like to thank Srin Rayanthanam and Yanchao Yu for their help and time integrating the User Model Component into the SpeechCity Android application and Andy Taylor for help on retrieving the data that I needed from the City Model.

Finally, I would like to thank family and friends who gave their time to evaluate the User Model component.
Abstract

The aim of this project is to develop a dynamic User Model and a method of populating the model for speech interfaces, in order to produce a personal assistant in the same vein as Apple’s Siri or Google’s Google Now. The system is able to use data retrieved from a City Model in conjunction with data on a user’s position and learn a user’s preferences, which are used to build a User Model, and recommend locations which a user will like. The system is also able to parse user utterances and look for any comments which point to a preference of the user and update their model.

The project looks at technologies such as User Modelling, spoken dialogue systems and data collection in order to give knowledge about what will be used during the development process. The design and implementation processes are detailed and an evaluation is presented contrasting use of the system with a default and populated User Model in order to judge the success of the final system.

Results of the evaluation show that users significantly prefer the dynamic User Model to a system with no User Model. The evaluation shows that the User Model integrated into the SpeechCity Android application is a desirable and useful feature, with the User Model receiving positive feedback, outperforming a system without a populated User Model.
# Table of Contents

Declaration ......................................................................................................................... I

Acknowledgments .............................................................................................................. II

Abstract .............................................................................................................................. III

Table of Contents ............................................................................................................... IV

List of Figures and Tables ................................................................................................. IX

  List of Figures ............................................................................................................... IX
  List of Tables ............................................................................................................... X

1 Introduction ..................................................................................................................... 1

  1.1 Motivation .................................................................................................................. 1

  1.2 Aims and Objectives ................................................................................................. 3
    1.2.1 Project Aim ......................................................................................................... 3
    1.2.2 Project objectives .............................................................................................. 4

  1.3 Worked Examples Information ................................................................................ 4

  1.4 Thesis Structure ........................................................................................................ 5

2 Literature Review ............................................................................................................. 7

  2.1 Overview .................................................................................................................... 7

  2.2 User Modelling .......................................................................................................... 7
    2.2.1 User Model characteristics ............................................................................... 8
    2.2.2 Use of a User Model .......................................................................................... 9
      2.2.2.1 User simulation .......................................................................................... 9
    2.2.3 User Model design patterns .............................................................................. 10
      2.2.3.1 Static User Models ................................................................................... 10
      2.2.3.2 Dynamic User Models ............................................................................. 11
      2.2.3.3 Highly adaptive User Models .................................................................. 11
      2.2.3.4 Stereotype User Models .......................................................................... 11
    2.2.4 Long term and short term User Models ............................................................. 12
    2.2.5 Modelling User Preferences ............................................................................. 12
10.2 What Has Been Achieved .................................................................................. 82
10.3 Limitations and Future Work ........................................................................... 82
10.4 Conclusions ...................................................................................................... 83
References ............................................................................................................. 85
Appendix A: User Testing Informed Consent Form.................................................. 89
Appendix B: Evaluation Tasks .................................................................................. 90
Appendix C: Pre-Use and Post-Use Survey .............................................................. 91
Appendix D: Parser Data XML File.......................................................................... 93
Appendix E: Default User Model XML File .............................................................. 98
Appendix F: Glossary ............................................................................................... 99
List of Figures and Tables

Below is the list of figures and tables contained in this report.

List of Figures

Figure 1: Actual and predicted growth of smartphone users from 2012 to 2017. (Emarketer.com, 2014) ______ 1
Figure 2: Example of the SpeechCity Android application. ________________________________ 2
Figure 3: ATOM dialogue engine. Shows the different components contained within a standard Spoken dialogue system (Agilingua.com, 2014). ________________________________ 15
Figure 4: Example of dialogue with the SpaceBook System (Janarthanam et al. 2012). __________________________ 17
Figure 5: Research into using User Models to improve a spoken dialogue system (Moore et al. 2004). ______ 21
Figure 6: The different User Models used to evaluate the system (Eckert et al. 1997). __________________________ 22
Figure 7: User Model for the exploration of a museum. The model is split up into different components (Sarini and Strapparava, 1998). ____________________________________________ 23
Figure 8: Iterative Waterfall model (Fcsittech.com, 2014). ________________________________ 28
Figure 9: Project Gantt chart __________________________________________________________ 37
Figure 10: SpeechCity system architecture. ____________________________________________ 41
Figure 11: SpeechCity system architecture after User Model component integration. _______ 41
Figure 12: Class diagram showing proposed classes, attributes and operations with their relationships ______ 42
Figure 13: City Model calls that will be used ______ 50
Figure 14: Example of a locations JSON retrieved from the City Model. ______________________ 50
Figure 15: Activity diagram of a user requesting a route to a location. ________________________ 55
Figure 16: Activity diagram showing the user utterance and its effect on the User Model __________ 56
Figure 17: Example of the LinkedList containing preference attributes. ______________________ 57
Figure 18: Example of the Hash Map used to represent the user’s preferences. ______________ 58
Figure 19: Example of original preference attributes. __________________________ 58
Figure 20: stateScore method. ________________________________________________________ 59
Figure 21: termScore method. _________________________________________________________ 59
Figure 22: Decrement formula. ________________________________________________________ 59
Figure 23: Assigning score based on decrement. __________________________________________ 59
Figure 24: Creation of the ordered map. _________________________________________________ 60
Figure 25: buildURL method to get URL to City Model ________________________________ 60
Figure 26: readAll method used to retrieve data from the City Model. ______________________ 61
Figure 27: readAll method used to format the City Model data into a JSON. ________________ 61
Figure 28: Code showing how the recommendations are generated. _______________________ 62
Figure 29: Creation of Hash Maps which store parser data. ____________________________________________ 63
Figure 30: parseNonAnswer method iterations to match keywords in user utterance. __________ 63
Figure 31: Updating the User Model based on level of interest. ____________________________ 64
Figure 32: Examples of the <state> elements in the parserData.xml file. _______________________ 65
Figure 33: Examples of <pref> elements in the parserData.xml file. _____________________________ 65
Figure 34: Example of coordinates and utterance sent to User Model component. ____________ 67
Figure 35: Example of the format of the output JSON sent to the Interaction Manager. _____________ 67
Figure 36: Example of the userDA JSON. ________________________________________________ 68
Figure 37: Complete input JSON received by the User Model component ______________________ 68
Figure 38: buildURL method used to retrieve JSON from City Model using Id. ________________ 69
Figure 39: Recommendations presented to the user in SpeechCity. __________________________ 70
List of Tables

Table 1: Showing dependencies between requirements. Read dependencies from top to bottom. 33
Table 2: Showing the importance of each requirement. 33
Table 3: Showing the possible risks that might arise during the project. 38
Table 4: Showing the possible risks that might arise during the project (continued). 39
Table 5: Examples of words and phrases for disliking a location. 52
Table 6: Examples of words and phrases for liking a location. 52
Table 7: Six point Likert scale used in the Pre-Use and Post-Use surveys. 73
Chapter 1

Introduction

1.1 Motivation

The exploration and navigation of a new city can be a hassle. For a long time people have had to rely on the use of physical maps, tour guides and the knowledge of the local population in order to get around. These days, with the increase of technology in this area such as online maps and route planning, with 1.75 billion smartphones in the world (Emarketer.com, 2014) and the availability of tablets with web access many people rely on the use of technology to aid them while out and about in an unfamiliar location.

Figure 1 shows the growth of smartphone users from the year 2012 through to the projected growth of 2017. These numbers show a greater number of smartphones being used each year. This indicates that the number of people with access to online maps and route planning will also increase.

![Smartphone Users - Actual and Predicted Growth](image)

Figure 1: Actual and predicted growth of smartphone users from 2012 to 2017. (Emarketer.com, 2014)

Spoken Dialogue systems are becoming increasingly popular and are being used throughout the world to augment and simplify the interaction between user and system. Many companies are now integrating Spoken Dialogue systems into smartphone and tablet applications, such as Google’s Google Now, Apple’s Siri and Microsoft’s Cortana, in order to provide the user with an alternate way
of interacting with their device. Google’s Google Now is able to create date reminders, restaurant reservations, call contacts, search the web, perform calculations, play music, etc. all by using speech.

The SpeechCity Android application is a Spoken Dialogue system which aims to provide a speech-driven, hands free, eyes free solution to exploration and navigation for pedestrians. SpeechCity aids users throughout their exploration of a city, allowing the user to request routes and listen to the history of certain locations. Figure 2 shows the SpeechCity application.

![SpeechCity Application](image)

**Figure 2: Example of the SpeechCity Android application.**

Personalised services are also becoming increasingly popular. Many organisations are now offering personalised services which aim to aid in the day to day life of its users. A personalised service allows a system to provide a more targeted delivery of its functionality to a user. This leads to a system which is easier to use than a generic one for all users. An example of this can be seen in Google’s Google Now (Google.co.uk, 2014). Google Now is able to learn the habits of a user; predicting where they will be at a certain time, giving information on journey times and recommend things the user may like such as new books, video games, albums and locations.

In order to provide a personalised system User Models are required. A User Model is able to build a picture of what a user is like. The model can be built up from any relevant information which can be used to describe the user. For example, information on the user’s personality, job titles, gender, age, information relating to any preferences that the user may have about a certain subject, etc. The model can then be used with a software system to offer a personalised service.
The current version of the SpeechCity application allows a user to speak to the system and receive information on the city they are currently in. This information can be directions or historical information, depending on the question which has been asked. The inclusion of a User Model (UM) component would allow information to be collected as the user explores and navigates a city. Over time a model of the user’s long term and short term preferences will be built. This model can then be used to recommend locations specific to that user.

1.2 Aims and Objectives
This section describes the overall aim of the project and the objectives which have to be completed. The aim of the project is given in detail along with an example of how the final system should work as well as a summary of the objectives that will have to be completed during the development of the system.

1.2.1 Project Aim
The aim of this project is to develop a User Modelling component which will be integrated into the SpeechCity Android application. The component will be able to implicitly collect data from a user in order to construct a User Model. As the user explores a city using SpeechCity the User Model component will be used to recommend locations a user is likely to enjoy. Finally the success of the system has to be evaluated.

The User Model component will implicitly collect information about the preferences of a user while they are exploring or travelling around a city using the SpeechCity Android application. Actions such as requesting a route to a location as well as the ability to listen for statements made by a user will update the User Model. For example, a user may say “I am interested in museums”, this will update the User Model to show that the user has an interest in museums or the user may say “Give me directions to Costa”, this will update the User Model to reflect that the user has shown an interest in Coffee Shops. The information contained in the model will then be used to suggest new places that the user may like to go to. The User Model will be constantly changing to best reflect the preferences of the user.

For example, if the user is constantly visiting a certain type of establishment, such as museums and pubs, then this information is stored within the User Model. The User Model will also listen to what the user says, updating the model based on what it has heard, e.g. if the user says “I have gone off beer” then the model will be updated to reflect the fact that the user may not want to be suggested bars and pubs. The model is then able to recommend museums which are located near the user’s current position.
1.2.2 Project objectives

In order for the final User Model Component to achieve the aim of this project several objectives will have to be completed. These objectives encompass the development of the software and the write-up of the thesis. The objectives are summarised below:

- **Define the User Model**: The User Model must be defined; this is what and how the user’s preferences are stored, taking into account short term and long term preferences.
- **Collect user information**: The collection of information should occur as the user navigates throughout the city, requests routes to locations and speaks to the SpeechCity App.
- **Build the User Model**: The User Model should be built using the collected information. It should also be able to be used across multiple sessions with the SpeechCity application.
- **Find nearby Locations to recommend using the User Model**: The User Model Component should be able to use the user’s GPS position and the City Model to generate recommendations based on the data contained in the User Model.
- **Relay recommendations to the user**: The User Model component should pass the generated recommendations to the user.
- **Distinguish long term from short term preferences**: The User Model should be able to represent long term and short term user preferences. Long term preferences will be those which the user feels most strongly about, e.g. favourite food and favourite places. Short term preferences will be those which the user shows an interest in, e.g. requesting routes to new places. Short term preferences will become long term preferences if the user continues to show an interest in them.
- **Update the User Model**: The data contained in the User Model should be dynamic so as to have a User Model which contains the most up to date information possible.
- **Integrate the User Model Component into the SpeechCity Android application**: The final system should communicate with SpeechCity’s Interaction Manager (IM) in order to receive the user’s utterances and position and to send the generated recommendations to be presented to the user.
- **Evaluate the success of User Model**: Find out how successful the User Model is when integrated into the SpeechCity application.

1.3 Worked Examples Information

In order to best describe the functionality of the User Model component worked examples will be used. The examples will be used throughout the thesis to describe the different parts of the system.
and demonstrate their functionality. The examples will be based on a user with the following information:

A user has the following preferences which will be used to build a User Model:

- Their favourite food is Italian.
- They love museums.
- They are interested in movies.
- They like video games.
- They hate art.
- They do not like seafood.

1.4 Thesis Structure

The overall structure of the thesis is organised into 10 chapters. The structure of the remaining chapters is as follows:

Chapter 2 – Literature review: A review of the current literature; this literature review focuses on the topics which are relevant to the development of the User Model, such as User Model creation, modelling user preferences, previous work in the subject, current systems and data collection. Academic journals, papers and books are used in order to show the current state of the field and to show any gaps in the literature which this project might fill.

Chapter 3 – Methodology: Discusses the methodology of the project along with the adopted software development methodology. Moreover, the evaluation method is presented.

Chapter 4 – Requirements specification: Presents the requirements specification. This defines the functional and non-functional requirements of the component detailing their importance and dependencies.

Chapter 5 – Professional, Legal, ethical and social issues: Describes the professional, legal, ethical and social issues which this project will face.

Chapter 6 – Project Plan: Shows the plan of the project, detailing the estimated time of completion for each task, and any risks which might arise during the course of the project.

Chapter 7 – Design: A detailed presentation of the design of the system. How the function and non-functional requirements were used in order to design the component and how the functionality of the final system will be implemented.
Chapter 8 – Implementation: A presentation of how the system was implemented showing how the design was utilised in order to produce the final system. Each function of the system has been described in order to show how it relates to the design and requirements.

Chapter 9 Evaluation: The evaluation of the system is presented. This chapter determines how successful the User Model Component is once integrated into the SpeechCity Android Application. An analysis of the results is presented as well as a conclusion on these results and the evaluation method.

Chapter 10 – Summary: Summary of the project as a whole, reviewing what was achieved and the current limitations of the implementation. Future work that could be undertaken, in order to improve the functionality of the component, is also discussed.

Finally references to work cited throughout and appendices of important files and screenshots are detailed at the end of the thesis.
Chapter 2

Literature Review

2.1 Overview

This literature review will focus on the topics which are important to the undertaking of this project. These topics are:

- **User Models and modelling**: Looking into what makes a User Model and the design patterns that have been used in previous modelling attempts.
- **Spoken dialogue systems**: With a look at the SpaceBook Android application, Google Now, Siri and Microsoft Cortana.
- **User Models in spoken dialogue systems**: Looking at any exploration applications which use User Models.
- **Implicit and explicit data collection**: Looking at the two types of data collection and how they are used.

Finally an analysis of the literature is given specifying what the focus of this project will be.

2.2 User Modelling

User Modelling is the creation of models of a user that can contain information on their interest and preferences which resides inside a computational environment. These models can be accessed or queried by a larger system in order to receive the information contained in the model.

User Modelling is currently an area of research which is receiving a lot of attention and propositions due to the need and benefits of including User Models within systems (Fischer 2001). The main goal of User Modelling is to create systems which are of benefit to the users using them.

In order for a system to interact with a user in an intelligent and cooperative way it has to know many things about the user that it is interacting with.

Examples of systems which rely on User Models in order to better aid the user are help and advice systems, tutorial systems and natural language systems (Abdel-Hafez and Yue 2013, Brusilovsky and Millán 2007, Fischer 2001).
2.2.1 User Model characteristics

There are many characteristics of a User Model which determines its lifespan, type, implementation and the information contained within it.

The information which is contained within a User Model can be used in many different ways. Kass and Finin (1998) tell us that there are four general uses for this information. This is further supported by Eberle (2011) in work done on User Models using stereotypes. Information received from the project supervisor reveals a fifth use. These five uses are:

1. To support the task of recognizing and interpreting the information seeking behaviour of a user.
2. To provide the user with help and advice.
3. To draw out information from a user.
4. To provide information to the user.
5. To adapt interaction style to the user.

Kass and Finin (1998) further explain that the information which is stored in a User Model can be divided into four different categories. These four categories are:

1. Goals and plans
2. Capabilities
3. Attitudes
4. Knowledge

However, the work done by Kass and Finin (1998) was solely on natural language systems, therefore, these four categories are heavily influenced by the system they were working on. Other systems may have different ways of using the data in the model.

There are two types of specialisation when it comes to User Models: generic and individual (Kass and Finin, 1998).

The generic User Model takes the view that all of the users that will be using the system are very similar and have similar goals when using the system. Therefore it can then treat each user as the same and have one model for all users.

The individual User Model takes the view that each user is an individual and keeps information based on a single user in a separate model. This is further reinforced by Eberle (2011), stating that the individual User Model provides a personalised view of the system to the user.
2.2.2 Use of a User Model

Kass and Finin (1998) tell us that a User Model can be used by a system in one of two ways. The first is descriptively and the second is pre-scriptively, with the first method of use being the most common way to use a User Model.

Descriptive use of a User Model allows the model to be used as a database containing the information collected on a user, a system has the ability to then query the User Model to find out what is stored on a user.

Pre-scriptive use of a User Model means that the model simulates the user for the benefit of the system (Kass and Finin, 1998). Pre-scriptive use of a User Model has been used in order to test systems without the need of actual users and is also called user simulation in spoken dialogue system research (Rieser and Lemon, 2011).

2.2.2.1 User simulation

User Modelling and User simulations are two different tasks. Both however, require and use a User Model.

Eckert et al. (1997) tells us that User Models are able to be used by a system to evaluate it. This is user simulation.

A series of User Models are populated with information about a series of users. These User Models can then be used pre-scriptively within the system.

The model acts as a user and the system runs as if the model were a user. This cuts out the need for a user to be present or testing and evaluation. This is done when it is too costly to have real users testing the system, or when it is necessary to test the system using user simulations before allowing real users access (Zhaojun et al. 2012).

This type of use of a User Model can be seen in the work done by Eckert et al. (1997). In order to test a spoken dialogue system there has to be a readily available large amount of real dialogue, which can be costly if real users are used. The purpose of the research carried out by Eckert et al. (1997) was to evaluate a system by using User Models alone, with no real users being used throughout the evaluation.

More on the use of User Models in spoken dialogue systems and the results of authors performing evaluations using User Models is given in section 2.3.
2.2.3 User Model design patterns

A review of the literature has revealed that the implementation of User Modelling can be split up into four different design patterns. These four patterns are:

- **Static**: Models where the contained information never changes.
- **Dynamic**: Models where the information changes to reflect a changing user.
- **Highly adaptive**: Models where a hybrid of static and dynamic models are used.
- **Stereotype**: Models information is assumed about a user based on a series of user classes.

These four User Models are explained in depth in sections 2.2.3.1, 2.2.3.2, 2.2.3.3 and 2.2.3.4 respectively.

The type of information that is to be stored by the model determines which design pattern should be used. A combination of the four User Models is usually preferred when building a User Model (Hothi and Hall, 1998). For example, this can be a combination of the static and dynamic models or the dynamic and stereotype models, etc.

Many User Model implementations in the past were designed using the static model. Today many implementations are being shifted from the static type of model, which store information such as gender and age, to a dynamic User Model (Hothi and Hall, 1998), which will typically use static elements of a model combined with the ability to alter data and add new data to a model.

### 2.2.3.1 Static User Models

Static User Models are models which contain information which cannot and will not be changed (Kass and Finin, 1998) throughout interaction of a user with a system. The information is entered into the model once and only once and remains constant throughout the use of a system.

The static model can either be pre-populated with information before the user can use it or it can acquire the information for the model based on the first interactions the user has with it (Kass and Finin, 1998). This is would mostly consist of answering questions in order to build the model, the system would then be able to start using the model as intended.

A combination of static and dynamic User Models can be used to produce a model which can be altered over time and contain static data which should not and will not change over the course of the session.
2.2.3.2 Dynamic User Models

Dynamic User Models are models which are able to change over time (Kass and Finin, 1998) and can model an up to date representation of a user. These models are able to constantly receive new information about a user which can be used to update an already built model. This allows changes in a user’s preferences to be taken into account.

Experiments carried out by Hothi and Hall (1998) have shown that users are more accepting of adaptive systems and that while a static model is useful the adaption of a system to a user through the use of a dynamic model was better able to aid the user throughout the use of the system. However, the experiment only took the static and dynamic User Models into account. There was no mention of the stereotype User Model and whether or not the system being used was utilising a long term or short term model.

Others have suggested that the static model is better suited for long term models (Kass and Finin, 1998) and that a combination of both the static and dynamic models provides a better implementation than using a static or dynamic model alone.

2.2.3.3 Highly adaptive User Models

The highly adaptive User Model is used when a model is needed of one user. It is different from the stereotype User Model in that it does not use classes of users to populate the initial model. This model requires a greater amount of information on a user before the model can be used.

2.2.3.4 Stereotype User Models

The stereotype User Model was first developed by Elaine Rich and is one of the oldest techniques for User Modelling (Brusilovsky and Millán 2007).

One way for a system to use its knowledge about the different classes of user using it and the information it has on an individual user is to use the stereotype User Model (Kass and Finin, 1998). This type of User Model allows a model to be built of a user in a very quick way (Rich 1989).

These classes are usually assigned to the user based on a series of questions on first use and the model of the user is then populated based on the class they fit into. An example of using classes is the flights system produced by Moore et al. (2004). This system used three User Models; Student, Frequent Flier and Business Class, one of which is assigned to a user after first use. An in depth look at this work is given in section 2.3.1.
Rich (1989) goes on to tell us that in order for a system to successfully use stereotypes to model a user it must have access to the stereotypes which it will apply to the users and it must have access to the events which occur which tell the system which stereotype the user belongs to.

Rich (1989) tells us that a stereotype, in regards to a User Model, is a body which contains information which is typically true about a user which the stereotype applies to.

Finin and Drager (1986) tell us that there are different ways to approach the stereotype User Model. One way is to allow the user to select which stereotype they fit best from a list of pre-defined stereotypes. Another way is to allow an expert system to ask the user questions and select a stereotype based on their answers.

Any changes in the user which affect which stereotype a user should be long to mean that the user will have to be re classified (Brusilovsky and Millán 2007).

2.2.4 Long term and short term User Models
The specialisation and type of User Model also determines if it is either long term or short term.

Long term models represent aspects of a user which will be required over a long period of time and are not going to be changed on a regular basis (Eberle et al. 2011). Long term models usually contain background knowledge and user preferences.

Long term User Models tend to be individual and use the static, dynamic or stereotype models, with short term models being generic and dynamic (Kass and Finin, 1998).

Short term models are usually used for one-off sessions and do not have a facility for remembering or storing information between sessions. Eberle (2011) tells us that short term User Models store data which is linked closely with a user and is needed only for completing the current objective of the system.

A combination of long and short term can be used. Eberle (2011) tells us that long term User Models can be used to populate an initial short term model.

2.2.5 Modelling User Preferences
There are many different ways to model a user’s preferences. Schickel-Zuber and Faltings (2006) tell us that there are two main ways to do this. The first is collaborative filtering, which takes a group of users and groups them together based on what they like. This technique assumes that users who are similar will like the same things (Schickel-Zuber and Faltings, 2006). The second method is explicit
preference gathering, where the user is explicitly asked about their preferences, which are then stored.

However, there are other ways in which preferences can be stored. H. Syed and Andritsos (2007) created a tree structure for storing user preferences as ontologies. Here each node of the tree represents a preference of the user and is related in some way to is adjoining nodes. Moving down the tree preferences become more specific, upward and the preferences are more general. H. Syed and Andritsos (2007) state that all users are different and should be understood by looking at information unique to that user; this could be their background, behaviour, preferences and interests.

H. Syed and Andritsos (2007) found that their tree structure was able to use the preferences to recommend papers which were relevant to the user. They also found that the more detail a user gave when specifying their preferences with the system the more accurate the recommendations became (H. Syed and Andritsos, 2007).

Research has been done in the field of learning user preferences based on sets. This is where a user sets of objects are compared based on whether the user likes them. This mainly takes the form of the user specifying that they prefer one set of objects to another (Eaton, n.d.). The drawback of this approach is that usually there are no attributes associated with the different objects in the sets. In order to capture how the user feels about different sets they have to explicitly declare it (Eaton, n.d.).

2.3 Spoken dialogue systems

Spoken dialogue systems are systems where a user is able to interact with the system using speech, with the reply from to the user’s interactions being spoken back to the user by the system. This type of system has the ability to interact with a user using speech instead of the more common place text system (Janarthanam et al., 2012, Williams et al. 2012, Chandramohan and Pietquin 2010, Lee et al. 2010).

A spoken dialogue system contains two components which are not included in a regular text system. These two components are a text-to-speech component and a speech recognizer component. The latter is used to recognise what the user is saying to the system and the former used to build a response to be spoken back to the user (Lee et al. 2010).

2.3.1 Common architecture of spoken dialogue systems

There are many different architectures when it comes to spoken dialogue systems. However, there are components that are usually common to all spoken dialogue systems (Lee et al. 2010). These components are:
- **A speech recognition engine**: This takes the speech which the user has given to the system and transforms it into something that the system can understand and work with. This is usually a word-by-word transcription of what has been said (Lee et al. 2010). This component consists of the acoustic word models and the language models in order to achieve its end result.

- **Acoustic word models**: Audio recordings and textual transcripts are combined in order to create statistical representations of the sounds that make up words. Acoustic word models are used by the speech recognition engine to recognise speech.

- **Language Models**: Tries to capture the language being spoken and predict the next words in a sequence of speech.

- **A natural language parser**: This component will then give meaning and understanding to the words or phrases that the system is trying to understand (Lee et al. 2010).

- **A dialogue manager**: This manager controls what will happen next based on the state of the dialogue currently in the system (Janarthanam et al., 2012, Williams et al. 2012, Chandramohan and Pietquin 2010, Lee et al. 2010).

- **A Natural Language generator**: Generates the language to be spoken back to the user from machine representations.

- **Text to speech**: Generates speech from text which is spoken back to the user.

Figure 3 shows the ATOM dialogue engine. This shows the standard architecture of a spoken dialogue system, showing the different components that are required by a spoken dialogue system and how they interact with one another in order to understand speech given to it and output speech to the user of the system.
2.3.2 Current spoken dialogue technology
There has been an increase in the work being done on spoken dialogue systems both in theory and in practise (Williams et al. 2012), much of the work being done is focusing on the mobile setting.

Williams et al. (2012) tells us that the increase in theoretical and practical research has led to many big organisations developing spoken dialogue systems on a massive scale, most notable is Apple’s Siri, Google Now and Microsoft Bing voice search. Many organisations are looking into implementing spoken dialogue systems in cars to increase safety while driving, e.g. Lenovo.

It is now a real possibility that a speech interface can replace the traditional keyboard input in the mobile setting. There is an increase in the acceptance and use of speech to search and operate a device (Williams et al. 2012).

Williams et al. (2012) suggests that the number of users of spoken dialogue systems and the large amount of readily available speech data is also a contributing factor to the increase of interest in the research community.

2.3.3 Speech Navigation
McNeill’s (2002) research into navigation in virtual environments using spoken dialogue systems has found that speech is a more natural form of human-computer interaction, allowing for hands free use of a system or as an extension to the regular text based systems. McNeill (2002) reveals that the research that has been done into using spoken dialogue technology within navigation systems was surprisingly sparse. This research was concerned with spoken dialogue technology being applied to
navigation within virtual environments, however, the conclusions that the author derived can also be applied to navigation in the real world.

Herbig et al. (2010a) conducted research into the detection of types of users using a spoken dialogue system and User Models with stereotype classes. This research concluded that spoken dialogue systems can be greatly improved by personalising the system to the user’s behaviour. The authors put forward that the inclusion of personalisation within a spoken dialogue system would increase the accuracy of speed recognition.

Having a system apply only one language model to all users causes a mismatch between the system’s speech pattern and the characteristics and habits of the speaker (Herbig et al. 2010a). It is suggested that data about a user could be stored in order to identify that user when they use the system. This data could be certain characteristics and habits in their speech, meaning that each user will have their own acoustic model and language model. This data could be stored in a User Model, with the system accessing the User Models in order to determine which user is currently speaking to the system.

This mismatch between the system’s speech pattern and the user’s speech is also noted in work carried out by López-Cózar et al. 2010. It is a well-known fact that the speech that a user gives to a system is often misheard, misunderstood, or unrecognised in a spoken dialogue system. This is partly due to the system being designed to apply a single language model to a large number of users.

The results of the research conducted by Herbig et al. (2010a) shows that it is possible to identify existing and new users of a spoken dialogue system, increasing the accuracy of speech recognition and providing a personalised system. This is also shown in later work done by Herbig et al (2010b). This further shows that a personalised system, tailored to a single user, provides a better user experience and increases the usability of a system.

A review of the literature shows the main applications of spoken dialogue systems. One main focus is in transport applications. This focus can be seen in multiple papers and journals showing that the use of spoken dialogue systems for use while driving has increased safety and security, allowing drivers to operate devices hands-free (Herbig et al. 2010b), (Herbig et al. 2010a) and (McNeill et al. 2002).

Work has been done using User Models with spoken dialogue systems. This has been either prescriptive (Zhaojun et al. 2012, Eckert et al. 1997) and descriptive (Moore et al. 2004, Kobsa 1990). This will be discussed in section 2.3.
2.3.4 SpaceBook

SpaceBook is a speech driven, hands free and eyes free device for exploration and navigation (Spacebook-project.eu, 2014). SpaceBook is a spoken dialogue system and is the larger application which the User Model component of this project will be integrated to.

Figure 4 shows an example of a user speaking with the SpaceBook system and receiving a response.

Figure 4: Example of dialogue with the SpaceBook System (Janarthanam et al. 2012).

A review of the literature reveals the systems architecture and the components which make up the SpaceBook device.

SpaceBook consists of several different components which are all used in order to direct the user to points of interest and to answer and receive questions from the user. The Interaction Manager is responsible for planning system output (Mackaness et al. 2014).

Automatic speech recognition and text-to-speech software are used in order to take any user speech that is given to the system and convert it into text, the text-to-speech component is then used to convert any responses that are returned from text to speech in order to deliver the response back to the user (Janarthanam et al. 2012).

SpaceBook holds a model of the city that the user is currently exploring. This model is managed by the city model component and represents a 3D model of the city, this is so that a field of vision can be built as to determine what is currently being seen by the user (Mackaness et al. 2014). SpaceBook also contains a pedestrian tracker which is used in order to give an accurate position of the user and also to keep a list of the locations that the user has been.

In order to answer any questions posed by the user a question answering component is included. This deals with the questions asked (Janarthanam et al. 2012).

Experiments carried out by Janarthanam et al. 2012 (2012) sought to evaluate SpaceBook by allowing users to test the system. The results from those experiments show that the users testing the system found the SpaceBook exploration aspect was preferred to that of Google’s. However, it is stated that
given the choice users preferred Google’s speech and navigation. This could be just that the users were more used to using Google’s services and that given more time they may have come to prefer SpaceBook.

There are applications and services which offer similar functionality to the SpaceBook project. The most notable of these are Apple’s Siri and Google Now. However both of these applications require the user to look at the screen in order to interact with it.

2.3.5 Google Now

Google Now is developed by Google to be an intelligent personal assistant and is available on the Android operating system as well as the chrome browser. Its main function is to learn information about a User in order to make recommendations, answer questions posed by the user and complete tasks given by the user (Google.com, 2014). Some of these task are:

- Sending emails and texts
- Making phone calls
- Creating calendar events
- Retrieving routes to locations

Google Now is able to learn where the user lives and works and will let the user know how long it will take, if any obstacles will impeded their journey and if there are any interesting locations on the way.

2.3.6 Apple’s Siri

Siri is Apple’s voice based personal assistant and pre-dates Google Now. Siri is available on Apple’s iOS making it available on all new phones produced by Apple. As with Google Now the user is able perform tasks by speaking (Apple.com, 2014). Some of these tasks are:

- Sending messages
- Scheduling meetings
- Making phone calls

Siri allows a user to find locations by asking for them. This could be in the format of “Find me a Chinese restaurant”, and Siri will recommend a restaurant for the user to visit.

There have been limitations to the voice recognition of Siri. Early versions had a hard time recognising speech uttered by users with strong accents, most notably the Scottish accent.
At the current time Apple is looking at integrating Siri with voice control systems in cars. This will allow an eyes free operation of the system; allowing drivers to play music, get directions and perform all of the tasks available to Siri users (Apple.com, 2014).

2.3.7 Microsoft Cortana

Microsoft Cortana is a mobile personal assistant, which uses natural language processing, available for windows phones running windows 8.1 that replaces the default search (Windowsphone.com, 2014). It is designed to manage tasks which will be encountered in everyday life. As with Google Now and Siri, Cortana is able to perform tasks which aid in the use the device such as:

- Making phone calls, texts and emails
- Set reminders
- Get information on interests
- Get directions and recommendations on locations
- Stop calls and texts during a set time

Cortana has been designed to learn about a user and store the important information for later use (rival, 2014).

A difference between Cortana and both Google Now and Siri is that the user is able to talk to Cortana as if it were a person. Cortana can tell jokes, give opinions and share stories (Windowsphone.com, 2014).

2.4 User Models within spoken dialogue systems

As the purpose of this project is to implement a User Model component into a spoken dialogue system the following section investigates what is currently available in this form.

User Models have been implemented into spoken dialogue systems in different ways.

2.4.1 Descriptive use

As stated earlier a User Model can be used as a “database” in order to query the model and receive the information it contains back. This is usually done to fulfil the requests of users to the system.

Moore et al. (2004) have conducted research into generating tailored, comparative descriptions in spoken dialogue. This involved using User Models within a spoken dialogue system.

This research involved storing flight preferences in the different User Models: Student, Frequent Flier and Business Class, and then applying those preferences when a user asks for details on a flight from one city to another. The user speaks to the system asking about a certain flight, the system then stores
the data needed in order to query a database. The system then uses the correct User Model in order to determine what the best possible flight is and presents it to the user (Moore et al. 2004). This is similar to the functionality which will be employed by the User Model component for this project, mainly the user will be presented with locations based on the information contained in the user’s model.

Figure 5 shows the different responses of the system to the user obtained by Moore et al. (2004) during their research into using User Models to improve a spoken dialogue system, this was based on who the user was and shows the difference in responses due to using a different User Model for each user.

The system assigns a User Model to a user based on one of three classes. The User Models used are Student (S), Frequent Flier (FF) and Business Class (BC). Answering questions on start allows the user to be assigned a class and a User Model. Depending on the class of the user the response from the system and the suggested flight is different. Figure 5 shows examples of this. For example, Figure 5 shows that a Student is offered the cheapest flight and is told the downside, e.g. it is a connecting flight. The Frequent Flier is shown many different options. Here the user is told about different flights and if there are any business class seats. This User Model allows the user to be told different flight, the downside for each and more information on what the flight entails. The Business Class User Model suggests flight where business class is available or direct flights where business class is not available. The cheapest and connecting flights are not suggested.
Figure 5: Research into using User Models to improve a spoken dialogue system (Moore et al. 2004).

The research here involved creating a User Model and allowing a user to rank the preferences which it contained. The user had to do this on first use when registering to use the flights system (Moore and Foster et al. 2004). This is one of the approaches taken during the development of the User Model for this project, allowing the user to specify initial data before using the system.

2.4.2 Pre-scriptive use

The User Model has also been used in order to evaluate a system using user simulations. This is done when a system needs to be tested and either it is too costly to have real users test the system or it is done to test the system before real users get access to it.

Research has been conducted by Eckert et al. (1997) on the evaluation of spoken dialogue systems using User Models. The evaluation conducted used a User Model to perform evaluation based on user simulations.

In order to evaluate a spoken dialogue system there has to be a large number of sample dialogues, without the use of User Models this would mean using real users, costing more time and is more prone to errors (Eckert et al. 1997).

For the evaluation of the system the users were sorted out into different levels of experience, which can be seen in Figure 6.
The findings reported by Eckert et al. (1997) show that using the different User Models in order to simulate users of the system was successful. The finding suggest that using User Models in order to test a spoken dialogue system is a real alternative to using real users.

### 2.4.3 User Models in exploration systems

The literature on User Models in exploration and navigation systems is very sparse. A single paper was found.

Sarini and Strapparava (1998) conducted research into building a User Model in order to aid the exploration of a museum. This is similar to the functionality of the component to be developed for this project.

The research looks into developing a system which is able to integrate real physical space with virtual space, producing an augmented space which aids in the exploration and information seeking tasks of the user. The system is able to provide the user with information based on the exhibits and also help the exploration of the museum, giving suggestions on what to go to next (Sarini and Strapparava, 1998).

The user is equipped with a hand held computer and headphones, infrared emitters are placed throughout the museum in order to provide the augmented space.

A User Model is used in order to store the user’s knowledge, interests and preferences (Sarini and Strapparava, 1998).

The model built for this system relies on the user answering initial questions to determine their knowledge, this is done before the system can be used. The authors have separated the User Model into three components (Sarini and Strapparava, 1998):

1. **The user’s knowledge:** This is gained from the initial questionnaire and stores facts of what the user has become aware of so far through use of the system (see Figure 7).
2. **The user's interests:** This is represented as activation/inhibition weighted network. The nodes of the network are ordered sets of entities which are the areas of the museum which the user is interested in. When a node is suggested by the system an activation impulse is sent to each of the nodes connected to that entity. The weights of the links are used to decrease the intensity of the impulse, getting weaker as it travels around the network (see Figure 7).

3. **The user's preferences:** The authors put forward that this models the preferences about what has been already shown to them, it could be either a weighted network, if it needs to be evolving, or a static model of facts (see Figure 7).

Figure 7 shows how the User Model was divided into the three components and how the facts are built and stored within the model.

![Figure 7: User Model for the exploration of a museum. The model is split up into different components (Sarini and Strapparava, 1998).](image)

The model is populated initially by use of a questionnaire. Once the user starts to look around and visit exhibits the model is updated in order to model the user’s interests and preferences, based on the behaviour of the user.

Sarini and Strapparava (1998) concluded that the dividing of the User Model into three components was helpful in planning what to show the user next and to avoid repetition of visits to the same location.

2.5 **Data Collection**

The gathering of data is a vital part of a User Model. The main purpose of a User Model is to acquire data on a user and model it, ready to be accessed by a larger application. The information contained within a User Model has to be relevant, accurate and plentiful, in order for requests from the larger
system to be fulfilled (Kass and Finin, 1988). Therefore the data gathering stage of building the model is crucial.

Data is usually gathered over time by observing and / or interacting with the user (Finin and Drager, 1986).

2.5.1 Implicit data collection

Implicit data gathering allows the system to monitor the behaviour or habits of a user and then inferring facts about the user based on what it has observed (Abdel-Hafez and Yue 2013).

2.5.2 Explicit data collection

Explicit data gathering allows the user to directly answer questions, usually on first use, in order to populate the model (Abdel-Hafez and Yue 2013). According to Kass and Finin (1998) explicitly gathering data is the easier of the two methods. However, Rich (1989) tells us that allowing the user to directly enter the initial data into the User Model is limited and ineffective, this is due to the fact that it may take an undesirable number of questions to acquire the necessary data, the user may not have the correct knowledge about what is being asked or the user may not want to answer the questions being asked (Rich 1989). This supports the opinion that it is better to gather the required data implicitly than explicitly.

This shows that there are problems when trying to collect data about a user. The fact that a User Model requires this data in order to function means that solutions to these problems must be found before the User Model can be built.

The developer has to determine which method of data gathering is more suited to needs of the system the model is being used in.

2.5.3 Implicit Personalisation

Implicit Personalisation is now being used by many companies, such as Google, Apple and Microsoft, in order to provide a more personal service for their users. This implicit personalisation learns information about a user through speech over time. This can be learning how your family is by using a system to send messages or phone calls. The system will store if the recipient is referred to as family, friend, etc. (Olson, 2014).

A system is able to learn new things about a user by monitoring what a user is saying and looking for speech which might reveal things about their likes and dislikes.

Using the information learned, a system will be able to tell the user about accidents which may hinder their journey to work, if there is going to be bad weather near their home and how long it will take to
get to your most visited locations (Google.com, 2014). These are all performed without any interaction from the user.

Implicit personalisation also raises concerns about the privacy the user. The personal assistant will have to share data between apps (Olson, 2014) in order to provide its full services and the user may not be able to restrict what data is being stored. There is also the problem that the company offering the personal assistant has access to this data, knowing where the user lives, works and where they will be at a certain time.

2.6 Analysis and conclusions

This review of the literature has shown the current technology when it comes to User Models, building and populating user modes and spoken dialogue systems. This has given a greater picture of what should be done when design the User Model component, which design patterns should be used based on the life of the model and the information which has to be stored in it.

The use of some combination of static and dynamic design patterns seems to be the most beneficial way of building the User Model for this project. This is because static elements are best used in order to model things which will not change about the user, this could be used in order to model preferences of the user which will not change. Since the model will have to be constantly updated and also include short and long term preferences the dynamic design pattern has to be used.

Given that there is enough time during the implementation of the project the stereotype design pattern can be used. This would be used to assign a user a class in order to initially populate the model. This stereotype class would then be able to be phased out as the model gains more information about a user, eventually discarding the stereotype completely after enough is known about a user.

Examples of User Models being used within spoken dialogue systems has a given an understanding on how User Models are used in those systems, either to evaluate or to be used as a database of user facts.

The lack of literature on User Models being used in exploration systems was surprising. The museum system produced by Sarini and Strapparava (1998) is quite similar to the User Model component which will be developed for this project, in that it uses a User Model to store user’s preferences and interests and directs them to locations based on that. However, the system produced in that work had to rely on the user interacting with a handheld computer, rather than an entirely spoken dialog interface.

Therefore, I feel that there is a gap in what has been produced in these fields. There are no applications which are entirely spoken, require no interaction with hands or eyes, which implement a User Model
in order to suggest new things the user may like to do based on their past behaviour and what they have said during use of the system.
Chapter 3

Methodology

The overall methodology which will be followed throughout this project in order to design, implement, integrate and evaluate the system is specified by the following steps:

- **Define the User Model**: The model will be defined showing how the User Model will represent the preferences of the user. The format of each preference and what attributes each will have to have, in order to best represent how a user feels about it, will also be defined.

- **Design and implement the User Model**: The User Model will be designed. This will be how it will store preferences and which data structures will be used.

- **Design and implement the parser**: The parser will be designed. The XML files that will be used to store keywords and City Model types and categories will be defined and populated. The process of matching keywords to the user utterance will be implemented. Moreover, parsing of route requests will be implemented.

- **Integrate the User Model with SpeechCity**: The User Model component will be integrated into the SpeechCity Android application. Code will be tailored to fit the requirements of SpeechCity, the process of receiving data from the Interaction Manager will be designed and implemented.

- **Evaluate the User Model Component**: Usability testing will be performed in order to judge how successful the implementation, and project, has been.

- **Reflection and redesign**: Feedback received from the usability testing will be analysed and the design and implementation of the component will be updated based on issues that the usability testing highlights.

3.1 Software Development Methodology

To best organise the time and workload of the project a software development methodology will be utilized. This will help to design, implement and test the software produced. The software development methodology that will be employed for this project will be the Iterative Waterfall method.
This methodology will allow the requirements to be defined, the system to be designed and then implemented. Due to the iterative nature of this methodology it will be possible to return to previous stages if any problems occur at the design and testing stages of the project. This will help immensely during the integration with SpeechCity as it is predicted that there will have to be some redesign and implementation needed in order to tailor the User Model component to the needs of SpeechCity and a format which the Interaction Manager can work with.

Integration with SpeechCity will occur during the implementation stage of the Waterfall method which will enable the returning to the design stage if needed, this will also allow the testing of the complete system in the testing stage.

3.2 Evaluation Methodology

In order to evaluate the success of the final project the software produced will have to be evaluated. The evaluation will try to find out how successful the User Model component is when integrated into the SpeechCity application, which will be done by performing usability testing. The number of users will depend on the availability of people ready to test the system, the best case would have 10 or more users testing the system and then answering questions on the performance.

Each participant will be required to complete a series of tasks and complete two surveys. These surveys will be done before and after the User Model has been constructed. The differences between the two surveys will then be analysed. The results of the analysis will show if the system is able to
learn a User’s preferences and recommend them suitable locations. The results will show if the User Model component is a desirable feature of SpeechCity.
Chapter 4

Requirements Analysis

4.1 Overview
This chapter describes the purpose and scope of the software to be produced for the final project. The functional and non-functional requirements of the software are presented, showing what the final software will have to achieve in order to be successful.

The functional and non-functional requirements for the User Model component are described here, detailing any dependencies and their level of importance. A glossary is available in appendix F to ensure that the reader understands any jargon that is used throughout the requirements specification.

The following is an example dialogue with the system in order to show the systems functionality. The example uses the outlined example information in chapter 1 section 1.3.

The user says “Give me directions to Game”. This utterance creates an action in the SpeechCity Interaction Manager. This action creates a JSON which is then passed to the User Model component. The user’s User Model is searched for the type of the location requested, in this case it is “Video Game Shop”. If this type is in the User Model then the model is updated, if the type is not in the User Model then it is added to the model.

4.2 Purpose
The SpeechCity Android Application allows for hands free and eyes free exploration of a new city. It is aimed at tourists looking to learn about a new city without the hassle of having to operate a device or reading information. Users have come to expect ease of use with newer technologies. The User Model component will allow information to be collected about a user, with the goal of being able to suggest new places that a user might like to go.

4.3 Scope
This software will operate while the user is out and about using the SpeechCity Android application. Information is implicitly collected on the preferences of a user by their speech and exploration of the city and a User Model is built. The information contained within the User Model can then be utilised to make recommendations to the user. The model will be dynamic; constantly being updated to reflect changes in the user’s preferences.
Since SpeechCity is an Android application the User Model component will be developed for the Android operating system. This means that the User Model component will be written in the Java programming language.

4.4 Requirements

The requirements for the User Model component are broken up into functional and non-functional. Each requirement is described in detail in the sections below. The dependencies and importance of each requirement are also presented in the form of diagrams.

4.4.1 Functional requirements

4.4.1.1 Communication with SpeechCity
The User Model component should communicate with SpeechCity Android application. More specifically it should send data to and receive data from the Interaction Manager. SpeechCity’s Interaction Manager has access to all of the information required in order to find the user’s location, receive user utterances and interact with the user.

4.4.1.2 Communication with the City Model
The User Model component should communicate with the City Model. This allows the User Model component to receive detailed information on the nearby locations, in order to generate a series of recommendations.

4.4.1.3 Implicit Data Collection
The User Model component should implicitly collect information on the preferences of the user. This enables a User Model to be constructed without directly asking for information from a user.

4.4.1.4 User Model Construction
The User Model component should construct a model of the user’s preferences using the information collected. This enables a complete and accurate picture of the user to be built allowing recommendations to be generated which will reflect their likes and dislikes.

4.4.1.5 Track Long Term and Short Term Preferences
The User Model component should keep track of long term and short term preferences and update these based on the users actions over the use of the system. This enables a User Model to reflect the most up to date information on a user.
4.4.1.6 Recommend Locations
The User Model component should recommend locations based on the information contained in the User Model which are appropriate and suitable for the user. Preferences in a User Model should be run through a series of rules in order to determine the most appropriate locations to recommend.

4.4.1.7 Persistent User Model
The User Model component should provide a way to make the User Model persistent. The User Model should be able to be utilised across multiple sessions.

4.4.2 Non-functional requirements

4.4.2.1 Programming Language
The software developed should be implemented using the Java programming language. This is to enable easy integration with the SpeechCity Android application.

4.4.2.2 User Modelling Design Pattern
The User Model constructed should use the dynamic User Modelling design pattern (see Chapter 2 section 2.2.3 for explanations of different design patterns). This allows the User Model to be updated as a user explores a city.
### 4.4.3 Requirement Dependencies
Each of the functional and non-functional requirements have dependencies. These dependencies make sure that any requirements that have dependencies cannot be implemented until the requirements that they depend on have been implemented. Table 1 shows the dependencies of each of the requirements.

**Table 1:** Showing dependencies between requirements. Read dependencies from top to bottom.

<table>
<thead>
<tr>
<th></th>
<th>4.4.1.1</th>
<th>4.4.1.2</th>
<th>4.4.1.3</th>
<th>4.4.1.4</th>
<th>4.4.1.5</th>
<th>4.4.1.6</th>
<th>4.4.1.7</th>
<th>4.4.2.1</th>
<th>4.4.2.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.1.1</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.1.2</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.1.3</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.1.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.1.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.1.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.2.1</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.4.2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

### 4.4.4 Requirement Importance
Each requirement has a level of importance, this is how critical the requirement is to the success of the software developed for the project. Table 2 shows the importance of each requirement. The possible values of importance can be Optional, Low, Medium, High, and Critical.

**Table 2:** Showing the importance of each requirement.

<table>
<thead>
<tr>
<th>Requirement Number</th>
<th>Requirement Name</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4.1.1</td>
<td>Communication With SpeechCity</td>
<td>Critical</td>
</tr>
<tr>
<td>4.4.1.2</td>
<td>Communication With the City Model</td>
<td>Critical</td>
</tr>
<tr>
<td>4.4.1.3</td>
<td>Implicit Data Collection</td>
<td>High</td>
</tr>
<tr>
<td>4.4.1.4</td>
<td>User Model Construction</td>
<td>Critical</td>
</tr>
<tr>
<td>4.4.1.5</td>
<td>Track Long Term and Short Term Preferences</td>
<td>Medium</td>
</tr>
<tr>
<td>4.4.1.6</td>
<td>Recommend Locations</td>
<td>Critical</td>
</tr>
<tr>
<td>4.4.1.7</td>
<td>Persistent User Model</td>
<td>High</td>
</tr>
<tr>
<td>4.4.2.1</td>
<td>Programming Language</td>
<td>Critical</td>
</tr>
<tr>
<td>4.4.2.2</td>
<td>User Modelling Design Pattern</td>
<td>Critical</td>
</tr>
</tbody>
</table>
Chapter 5

Professional, Legal, Ethical and Social Issues

5.1 Overview
This chapter details the professional, legal, ethical and social issues which may be encountered during the undertaking of this project.

5.2 Professional issues
Throughout the course this project there will be many references to work done by others. This will mostly be in the form of academic papers, journals and books. The project will require the study and use of similar systems to gauge how the software for this project will be built, integrated into a spoken dialogue system and how successful the end result is.

Any work used throughout this project that has been done by others will be fully acknowledged with the proper citations and references and full credit given to the original authors.

The system to be produced for this project is an extension of the SpeechCity Android application, this means that access to the application will be needed. Access to the SpeechCity Android application, for work to be done on the User Model component, will be given by the project supervisor and SpeechCity developers.

5.3 Legal, ethical and social issues
There are several legal and ethical issues which need to be looked at. These legal issues are concerned with data privacy. In order for SpeechCity and the User Model component to function as intended data about the user has to be collected and stored. A new user must register with SpeechCity, giving their name and email address.

In order for the User Model component to function data on the user’s preferences must be stored. The User Model component has access to the user’s location and builds a model of what the user is like, storing information on their navigation preferences. This data is stored on the user’s device allowing them full access to the information that is being collected. This data is stored anonymously meaning there is no possible way to link the stored data and a user and no personal information can be extracted from the User Model data if intercepted by unauthorised persons.
Further ethical issues arise during the evaluation stage of the project. The evaluation of the project relies on allowing users to test the final system. The evaluation experiments will take the form of tasks and surveys in order to receive a good amount of feedback on the finished software.

In order to resolve these issues testers will have to sign a user testing informed consent form informing them that the feedback they are giving will be stored anonymously and no connection will be made between the tester and any data that is collected during the testing stage.
Chapter 6

Project Plan

6.1 Overview

This Chapter shows the proposed work plan for the project and the risks which might arise during the development of the software and the write-up of the thesis.

6.2 Gantt chart

The following Gantt chart shows the timeline of the project. Each task which must be completed in order to produce the final software is shown, along with milestones which show the important dates, such as code completion, testing, evaluation and final hand-ins. The project has been split up into different stages to better help organise the workload. These stages are:

- **Design**: Designing the software to be produced for the project.
- **Implementation**: Implementing the design of the software and integrating the system with SpeechCity.
- **Testing**: Testing the finished system.
- **Evaluation**: Performing usability testing in order to determine the success of the software.

Write up of the thesis is done in parallel with the testing and evaluation stages. This helps to balance the workload near the end of the project.
<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Predecessors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>User model component</td>
<td>73 days</td>
<td>Mon 12/05/14</td>
<td>Thu 21/08/14</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Design</td>
<td>7 days</td>
<td>Mon 12/05/14</td>
<td>Tue 20/05/14</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Initial design of software</td>
<td>3 days</td>
<td>Mon 12/05/14</td>
<td>Wed 14/05/14</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Feedback from supervisor</td>
<td>1 day</td>
<td>Thu 15/05/14</td>
<td>Thu 15/05/14</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Finalise design</td>
<td>3 days</td>
<td>Fri 16/05/14</td>
<td>Tue 20/05/14</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Design complete</td>
<td>0 days</td>
<td>Tue 20/05/14</td>
<td>Tue 20/05/14</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>Implementation</td>
<td>32 days</td>
<td>Wed 21/05/14</td>
<td>Thu 03/07/14</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Implementation</td>
<td>20 days</td>
<td>Wed 21/05/14</td>
<td>Thu 17/06/14</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>Prototype of component</td>
<td>0 days</td>
<td>Tue 17/06/14</td>
<td>Tue 17/06/14</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>Finalize implementation</td>
<td>12 days</td>
<td>Wed 18/06/14</td>
<td>Thu 03/07/14</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>Implementation complete</td>
<td>0 days</td>
<td>Thu 03/07/14</td>
<td>Thu 03/07/14</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>Testing</td>
<td>10 days</td>
<td>Fri 04/07/14</td>
<td>Thu 17/07/14</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Integration with SpeechCity</td>
<td>10 days</td>
<td>Fri 04/07/14</td>
<td>Thu 17/07/14</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>Testing</td>
<td>10 days</td>
<td>Fri 04/07/14</td>
<td>Thu 17/07/14</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>Testing complete</td>
<td>0 days</td>
<td>Thu 17/07/14</td>
<td>Thu 17/07/14</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>Evaluation</td>
<td>14 days</td>
<td>Fri 18/07/14</td>
<td>Wed 06/08/14</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Perform user testing</td>
<td>12 days</td>
<td>Fri 18/07/14</td>
<td>Mon 04/08/14</td>
<td>14</td>
</tr>
<tr>
<td>17</td>
<td>Results and analysis</td>
<td>2 days</td>
<td>Tue 05/08/14</td>
<td>Wed 06/08/14</td>
<td>16</td>
</tr>
<tr>
<td>18</td>
<td>Evaluation complete</td>
<td>0 days</td>
<td>Wed 06/08/14</td>
<td>Wed 06/08/14</td>
<td>17</td>
</tr>
<tr>
<td>19</td>
<td>Write up thesis</td>
<td>31 days</td>
<td>Mon 07/07/14</td>
<td>Mon 18/08/14</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Final hand-in of project</td>
<td>0 days</td>
<td>Thu 21/08/14</td>
<td>Thu 21/08/14</td>
<td>19</td>
</tr>
</tbody>
</table>

![Gantt Chart](image)

**Figure 9: Project Gantt chart**
6.3 Risk evaluation

As with any project there are always going to be risks. These are the risks that can potentially come up during any stage of the project. Table 1 and table 2 show the risks that have been considered which have the potential to arise during the course of the project. The table is split into 6 columns. The six columns and their possible values are:

1. A description of the risk that may occur.
2. The type of the described risk. The different values are project or business.
3. The probability of the risk occurring. The possible values are low, medium or high.
4. The impact of the risk on the project. The possible values are low, medium or high.
5. Any indicators that the risk is going to occur.
6. The strategy for dealing with the risk if it arises.

Table 3: Showing the possible risks that might arise during the project.

<table>
<thead>
<tr>
<th>Description</th>
<th>Type</th>
<th>Probability</th>
<th>Impact</th>
<th>Indicators</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing deadlines</td>
<td>Project</td>
<td>Medium</td>
<td>High</td>
<td>Work not being completed by the time given by the plan</td>
<td>Regular meetings with supervisor and monitoring of tasks and deadlines</td>
</tr>
<tr>
<td>Exceed project completion time</td>
<td>Project</td>
<td>Low</td>
<td>High</td>
<td>Multiple deadlines missed and work not being done</td>
<td>Prioritise tasks which have fallen behind and monitor task completion times</td>
</tr>
<tr>
<td>Supervisor cannot be contacted due to other engagements</td>
<td>Business</td>
<td>Low</td>
<td>Medium</td>
<td>Communication with supervisor via emails</td>
<td>Keep regular contact with supervisor</td>
</tr>
<tr>
<td>Missing meetings with supervisor</td>
<td>Business</td>
<td>Medium</td>
<td>Medium</td>
<td>Communication with supervisor</td>
<td>Keep regular meetings and communicate with supervisor</td>
</tr>
</tbody>
</table>
Table 4: Showing the possible risks that might arise during the project (continued).

<table>
<thead>
<tr>
<th>Description</th>
<th>Type</th>
<th>Probability</th>
<th>Impact</th>
<th>Indicators</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration with SpeechCity encounters problems</td>
<td>Project</td>
<td>Low</td>
<td>High</td>
<td>Results during testing and implementation</td>
<td>Keep SpeechCity in mind during implementation. Make sure to thoroughly test UM with SpeechCity.</td>
</tr>
<tr>
<td>Evaluation reveals UM is not successful</td>
<td>Project</td>
<td>Low</td>
<td>Medium</td>
<td>Results from usability testing</td>
<td>The purpose is to design the UM and test to see if it a desirable feature. If it is not successful this is still a result and can be written about.</td>
</tr>
<tr>
<td>Cannot implement some functionality</td>
<td>Project</td>
<td>Low</td>
<td>High</td>
<td>Work not being done in implementation stage</td>
<td>Meetings with supervisor and regularly checking progress.</td>
</tr>
<tr>
<td>Failure to capture all requirements</td>
<td>Project</td>
<td>Medium</td>
<td>Medium</td>
<td>Feedback during testing and evaluation</td>
<td>Using an iterative development method to allow revision of requirements.</td>
</tr>
<tr>
<td>City Model is inaccessible</td>
<td>Project</td>
<td>Medium</td>
<td>High</td>
<td>City Model down</td>
<td>Contact supervisor / City Model developer.</td>
</tr>
<tr>
<td>Lack of research material</td>
<td>Project</td>
<td>Low</td>
<td>Medium</td>
<td>Unable to find the needed papers and journals</td>
<td>Widen search parameters and make better use of materials found.</td>
</tr>
<tr>
<td>Not enough users for evaluation</td>
<td>Project</td>
<td>Medium</td>
<td>High</td>
<td>Not enough questionnaires and feedback being received</td>
<td>Have family and friends join the usability testing.</td>
</tr>
</tbody>
</table>
Chapter 7

Design

7.1 Overview

The User Model component should be designed with efficiency in mind. This is so the recommendation calculations that are to be performed can occur at a regular interval, without slowing down or halting the normal functionality of the SpeechCity Android application. This chapter details the design of the software produced in order to achieve the aim and fulfil each of the requirements of the project. The functional and non-functional requirements are used to formulate a design that will accomplish the objectives and satisfy the requirements.

7.2 Architecture

7.2.1 SpeechCity Architecture

SpeechCity is composed of a series of components, which when communicate and run together allow the application to perform its functionality. These components allow the user’s position to be tracked, any utterances to be parsed, the prediction of what a user is able to see in front of them and also pass information to the user on specific locations. These components are:

1. **Interaction manager (IM):** This allows information to be passed to and received from the user. This could be when a user requests directions or the history behind a location.
2. **City model (CM):** The City Model is an accurate model of the current city. This is Edinburgh throughout the course of this project.
3. **Pedestrian tracker (PT):** This allows the user’s position to be tracked.
4. **Parser:** This is used to parse any utterances that the user has said and transform them into a form that is recognisable to the Interaction Manager.

Figure 10 shows the architecture of SpeechCity prior to integrating the User Model component (Janarthanam et al. 2012).
The User Model component being developed for this project will take the form of another component that can be communicated with via the Interaction Manager. Data will be sent to the User Model component from the Interaction manager and vice versa. This data will include the user’s position, any utterances and any route choices. As well as communicating with the Interaction Manager the User Model Component will also have access to the City Model. This allows the component to make its own calls to the City Model independent of the Interaction Manager.

Figure 11 shows the updated architecture of SpeechCity to include the User Model component and the data that is being sent to and from the Interaction manager and the City Model.

7.2.2 User Model component Structure
In order to design a system that achieves all of the requirements and meets the objectives multiple classes have to be used. Eight classes are used in order to separate the functionality of the system and at the same time reduce complexity. The classes are designed to make the system more modular.

Figure 12 proposes a class diagram of the User Model component. This shows all proposed attributes and methods of each class and their relationships.
Figure 12: Class diagram showing proposed classes, attributes and operations with their relationships
The functionality of the eight classes within the system is detailed below:

1. **Engine**: The engine class would hold the main method and any data needing to be passed to the UserModelComponent class.

2. **UserModelComponent**: This class will receive data from other classes in order to update the User Model. Moreover, this class will make calls to the Rules class in order to get the recommendations to be sent to the Interaction Manager and use methods in the Parser class to parse any user utterances. This class will do all of the calculations needed to generate the recommendations.

3. **Rules**: This class will hold all rules which can be applied to a location to determine its suitability. Any recommendation and update rules needed by the system will be held here. This class will be implemented as a static class which will allow calls to be made to it without any initialisation.

4. **Parser**: This class could hold methods which manipulate any user Utterances. It is planned to hold keywords and their respective City Model types here.

5. **JSONReader**: As the City Model returns data in the form of a JSON a class must be designed which allows their retrieval. This class makes the calls to the City Model and stores the results in the form of a JSON, ready to be used by the UserModelComponent class.

6. **FileIO**: This class will deal with reading and writing to any files that are needed or updated by the system.

7. **UserModel**: A class is needed which stores the user’s User Model. This class will provide methods for manipulating the contents of a User Model as well as storing the model itself.

8. **Preference**: The preference class will hold all information relating to a preference. This allows the UserModel class to store a preference and manipulate it with ease.

The eight classes will be separated into four packages. This will allow all similar classes to be grouped together increasing readability and maintainability. These four packages will be:

1. **umc**: This will hold the UserModelComponent and Rules classes.
2. **umc.main**: This will hold the engine class.
3. **umc.io**: This will hold the FileIO, JSONReader and Parser classes.
4. **umc.model**: This will hold the UserModel and Preference classes.
7.3 User Model Representation

The preferences of a user make up a User Model. The representation of the User Model has to be designed to include all possible information to best describe that preference and how a user feels about it.

The User Model will be designed so that the City Model type; e.g. café, shop, etc., or category; e.g. Bookstore, Coffee Shop, Bike Shop, etc., will be stored. In order to best describe each type in the User Model a preference object will be associated with it. This preference object will hold all information relating to the type it is associated with. The format of the entries in the User Model will be as follows:

\[ T : X, Y, Z \quad \text{where} \quad T = \text{City Model type or category} \]
\[ X = \text{State} \]
\[ Y = \text{Term} \]
\[ Z = \text{count} \]

Example: If we go back to the example user described in chapter 1 section 1.3 the preferences stated would build a User Model which looks like the following:

- Italian Restaurant : like, long term, 75
- Museums : like, short term, 50
- Movie Theater : like, short term, 50
- Video Game Store : like, short term, 25
- Art Gallery : dislike, long term, -75
- Seafood : dislike, short term, -25

The User Model will be a list of \( <T, X, Y, Z> \).

The following section describes in detail how the preference objects will be designed and what kind of information will be stored.

7.3.1 Preferences

Preferences make up a User Model. In order to best represent a user’s preferences each preference has to have a series of attributes that will describe the users’ preference. These attributes will then be used by the rules (described in section 7.4) to determine the suitability of a location that is to be recommended to the user. These attributes will be:

- **State:** \( X \in \{\text{like, dislike}\} \)
- **Term:** \( Y \in \{\text{short term, long term}\} \)
- **Count:** \( \{Z \in \mathbb{R} : Z \geq -100 \text{ and } Z \leq 100\} \)
7.3.1.1 Long Term and Short Term
Each preference in a User Model will be either a long term or short term preference. Every preference added to a User Model by requesting a route will be added as a short term preference. Long term preferences will be dependent on two things.

The first is how the user speaks about a preference. For example, if a user states that their favourite food is Italian then the City Model category **Italian Restaurant** is made a long term preference in the model.

The second is how many times a user requests routes to a location. If a user visits a location many times it will eventually become a long term preference. This is to show that the user has been constantly visiting a location. For example, if a user visits a coffee shop every day then the City Model category **Coffee Shop** will be updated every day. Eventually **Coffee Shop** will become a long term preference, this is based on the preferences count (see chapter 8 section 8.2.3.2).

7.3.1.2 Count
The count will reflect the level of like or dislike of a preference. Each preference’s count will be kept on a scale of -100 to 100. Anything less than 0 will be a dislike and anything greater than -1 will be a like. A preference with a count of 100 will be more liked than a preference with a count of 10.

The count will also incremented if a User chooses to go to a location of that type, however, the count will also be initially set higher for expressed liked preferences through any user utterances. This means that if a user where to say “I love art” then the category “Art Gallery” will be added to the User Model with a count of 50. The Parser Data XML file holds the difference levels at which the count can be set based on a keyword and can be found in appendix D.

7.3.2 Storing the User Model
Allowing a user to use the system and exit without losing the User Model that had been built is an important function of the system, without this the system would have to start fresh each time the SpeechCity application is started. This means that a user could start the application, after exiting, and still receive recommendations based on their preferences from previous sessions. A XML file will be used which will hold the User Model data, allowing the data to be read back into the system across multiple sessions.

The User Model XML file will be read into the system when it first starts up. Each time an update or an addition is made to the User Model it will be re-written to the XML file. This allows us to have an up to date version of the User Model even if the system crashes or something goes wrong.
More on the design of this XML file can be found in section 7.7.1.

7.4 Rules
The main purpose of the User Model component is to recommend locations to a user based on their preferences. In order to do this a series of rules has to be designed which would allow each preference, contained in the User Model, to be evaluated to determine the likelihood that a user would like it and to present a user with the most appropriate locations to visit. Therefore, the rules of the system play an important role.

The rules of the component will be split into two sets, recommendation rules and update rules. They will be designed so that as a preference in the User Model is put through each of the recommendation rules it is assigned a score, this score can then be used to recommend the most appropriate locations to the users.

The following subsections discuss the design of the recommendations and update rules.

7.4.1 Recommendation Rules
The recommendation rules will be used to assign a score to a preference type, which can then be compared against other preference type scores to recommend the most suitable locations to a user. There will be several different rules which can be applied to a preference type, all of which are used and make up the recommendation rules of the system. The implementation of these rules can be seen in Chapter 8 section 8.2.3.1.

In order to describe the design of the recommendation rules the following variables will be used to refer to the different scores applied by the rules which make up the recommendation rules.

- State rule: $s$
- Term rule: $t$
- Count rule: $c$
- Distance rule: $d$

Each of the attributes of a preferences will be assigned a score. These scores will then be added together in order to get the preference’s overall score. The overall score will be a simple addition and will be calculated like so:

$$\text{Score}(T) = s + t + c$$

Once each preference in the User Model has been assigned a score and the data has been retrieved form the City Model we will apply a distance rule, which will give the closest location to the user.
The following sub-sections describe the design of each rule that will make up the recommendation rules of the User Model component.

7.4.1.1 State Rule
The state rule will apply a score based on what the state of a preference is. If the preference is a “like” then it will receive a score of 1.0. In order to ensure that preferences which are dislikes of the user we will assign a score of -100.0, this negative score will be used by the system to filter out any dislikes that the User Model contains. To summarise:

\[
\begin{align*}
    s & \in \{-100, 1\} \\
    s: & S = \{\text{like}\} \text{ then } s = \{1.0\} \\
    & S = \{\text{dislike}\} \text{ then } s = \{-100.0\}
\end{align*}
\]

7.4.1.2 Term Rule
The term rule will apply a score based on the term of the preference, this is whether the preference is short term or long term. With this rule we will have to also to look at the state of the preference as we cannot give the same bonus to a long term “like” and a long term “dislike”. Therefore, checks will be made before any score is assigned. The following describes the term rule:

\[
\begin{align*}
    t & \in \{-100, -50, 1, 2.5\} \\
    t: & S = \{\text{like}\} \text{ and } T = \{\text{long term}\} \text{ then } t = \{2.5\} \\
    & S = \{\text{like}\} \text{ and } T = \{\text{short term}\} \text{ then } t = \{1.0\} \\
    & S = \{\text{dislike}\} \text{ and } T = \{\text{short term}\} \text{ then } t = \{-50.0\} \\
    & S = \{\text{dislike}\} \text{ and } T = \{\text{long term}\} \text{ then } t = \{-100.0\}
\end{align*}
\]

7.4.1.3 Count Rule
The count rule will work in a different way to the state and term rules. This rule will first organise all of the preferences in the User Model based on the value of their count attribute, with the preference with the highest count being first in the list. Each preference will then be assigned a score based on their position in the list. This will be done by having a hard coded maximum score value. The first preference in the list will be assigned the maximum value. Any dislikes will not be assigned a score. The remaining preferences will be assigned a score based on the following two formulas, where \(d = \) decrement, \(m = \) maximum score, \(n = \) number of preferences in list, \(\text{score} = \) score to be applied to current preference.

\[
\begin{align*}
    d &= m / n \\
    \text{score} &= m - d
\end{align*}
\]

Using these two formulas will allow each preference to be assigned a score based on how much a user has visited that location or spoken about a type of location.
**Example:** For the following example we will use the information from chapter 1 section 1.3. We will use one like and one dislike. More specifically we will use the two preferences:

- Favourite food is Italian.
- Do not like seafood.

The two preferences take the form:

- Italian Restaurant : like, long term, 75
- Seafood : dislike, short term, -25

**Italian Restaurant:**

\[
\begin{align*}
  s &= 1.0, \ t= 2.5, \ c = 100 \text{ (first in list)} \\
  \text{Score(Italian Restaurant)} &= s + t + c \\
  \text{Score(Italian Restaurant)} &= 1.0 + 2.5 + 100 \\
  \text{Score(Italian Restaurant)} &= 103.5
\end{align*}
\]

**Seafood:**

\[
\begin{align*}
  s &= -100, \ t = -50, \ c = 0 \text{ (since it is a dislike)} \\
  \text{Score(Seafood)} &= s + t + c \\
  \text{Score(Seafood)} &= -100 + -50 \\
  \text{Score(Seafood)} &= -150
\end{align*}
\]

For the calculations above we can see that the preference “Italian Restaurant” is the highest scored and will be recommended to the user.

7.4.1.4 **Distance rule**

In order to provide the most suitable recommendations to a user we will have to take the distance to a location into account. This ensures that the user will be recommended a location which is close to them. For example, if a user likes coffee shops they may be recommended Costa which is 400 meters from them even though there is a Starbucks only 50 meters from them. This is due to the JSON retrieved from the City Model being unordered. To solve this we will use the distance rule.

The distance rule will be applied after each preference has been assigned a score and the two preferences with the highest scores have been retrieved from the City Model. The distance rule will then take each location from both preferences and calculate how far it is from the user. The location closest to the user from each of the two preferences will be the ones that are recommended. This will be done using the following pseudo code:

\[
\text{currentPreference} = \text{preference}
\]
for all preferences p in User Model
if currentPreference distance > p distance
    currentPreference = p

Using the example above “Italian Restaurant” was the highest scored preference. This preference might have the following three locations with their distances from the user:

- Vittoria : 100m
- Domenicos : 50m
- Giulianos : 630m

The distance rule would then determine that Domenicos should be recommended to the user as it is the closest.

7.4.1.5 Score Rule
We do not want to recommend locations to a User if the user does not like them. In order to eliminate these locations we will use the score rule. The score rule will be applied after all other rules. This rule will look for preferences which have received a negative score. If there has been a negative score it will increment a count. This count will then be used to eliminate the disliked preferences. This is done by stopping the recommendation loop before it gets to the dislikes. For example, a list of preferences has 5 likes and 3 dislikes for a total of eight. The score rule will calculate that there are 3 dislikes. The following calculation is applied to the loop in order to stop it before any dislikes are recommended:

\[
\text{Size of list} - \text{score rule value} = \text{number of times to loop}
\]

In the case of the above example this would be:

\[
8 - 3 = 5
\]

So the recommendation loop will only loop over only five preferences, missing out the three dislikes.

7.4.2 Update Rules
As the User Model is dynamic there must be some way of updating its contents. This is where the update rules come in. These rules will be designed to work in conjunction with the parser (see section 7.6), in that they are used to update elements of the User Model if a user requests a route to a location or mentions a preference in an utterance. Moreover, the update rules are used to manage the long term and short term attributes and to ensure preferences have the proper state.

The update rules will be designed to check the count of the preference and make sure that it has the proper term and state. These rules will also be used to insert new preferences to the User Model.
An update rule will also be designed to manage the count of each preference. Once a location has been requested the rule will be used and the preference’s count will be incremented.

The implementation of these rules can be seen in Chapter 8 section 8.2.3.2.

7.5 Retrieving Data from the City Model

The City Model holds all information on locations in the City of Edinburgh. The main method of retrieving this information is through the use of calls to the city model. Each call returns a JSON of all matching entities. The User Model component is designed to work with the City Model by utilizing two of the available calls. Figure 13 shows the two different calls that will be used by the User Model component in order to retrieve the location data.

```
getNearbyEntitiesInfoByType
getEntityInfoById
```

Figure 13: City Model calls that will be used.

As stated above calls to the City Model return a JSON containing all of the matching entities. Each entity is in itself another JSON, this will allow us to iterate of the JSON of matching entities one by one.

7.5.1 City Model Types and Categories

The City Model stores information on each location, such as a locations name, coordinates, address, etc. Figure 14 shows the format of a locations data retrieved from the City Model.

```
"name": "Caffe Nero",
"id": "1106076816",
"OSM type": "node",
"position": "-3.2082848 55.9578836",
"type": "cafe",
"foursquare rating": {
   "category": "Coffee+Shop",
   "city": "Edinburgh",
   "telephone": "42B44+31+225+7226",
   "FoursquareValidity": "3",
   "postcode": "EH3+6SZ",
   "address": "4-6+Glenville+Pl",
   "checkins": "220",
   "icon": "https://3a%2F%2Fss1.4sqi.net%2Fimg%2Fcategories_v2%2Ffood%2Fcoffeeshop.png"
}
```

Figure 14: Example of the format of a locations JSON retrieved from the City Model.

The User Model component will require two things from a locations JSON. These are the type and category. Both of these are needed. Not every location has a category in the City model, if a Location
has a category it will be used to update the User Model. If a location does not have a category the
type will be used to update the User Model.

The categories in the City Model are more descriptive than the types and thus will make for a more
varied User Model. Some examples of City Model categories are, Italian restaurant, coffee shop,
bookstore, etc. See appendix D for a list of City Model categories.

As shown above calls to the City Model require a type, this type is used to search for all locations of
that type near the user. A location in the City Model will always have a type but may not have a
category. In the case where a category is not present then the type will be used. Within the City Model
there are five functional types, these types are:

1. Café
2. Shop
3. Restaurant
4. Bar
5. Named Building

A User Model with only these five types would not be very useful as a user may like Italian restaurants
but hate seafood; a User Model with only the above five types could not reflect this. In order to
produce a more varied User Model, that can better reflect a user’s preferences, City Model categories
will be used.

### 7.6 Updating the User Model

As this project requires a dynamic User Model there must a process of updating a User Model during
the use of the system. This must process must allow new preferences to be added as well as allowing
known preferences to be changed. The following subsections describe the design of the parser
functionality of the User Model component.
7.6.1 Parsing User utterances

In order to parse any user utterances SpeechCity will pass the String to the User Model component. This string will then be checked against keywords and phrases stored in an external XML file, more on the XML file can be found in section 7.7.2. If a match is found between the utterance and any of the keywords then the User Model will be updated, using the update rules. Some examples of the words and phrases are:

<table>
<thead>
<tr>
<th>Common Words and Phrases for liking a location</th>
<th>Common Words and Phrases for disliking a location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Favourite food</td>
<td>Hate</td>
</tr>
<tr>
<td>Love</td>
<td>Not interested in</td>
</tr>
<tr>
<td>Interested in</td>
<td>Loath</td>
</tr>
<tr>
<td>Adore</td>
<td>Detest</td>
</tr>
<tr>
<td>enjoy</td>
<td>Can’t stand</td>
</tr>
</tbody>
</table>

A full list of the words and phrases that are looked for by the parser can be found in appendix D.

7.6.2 Requesting Routes

Once a user has received their recommendations they will be able to view and request a route to them. This will involve the user speaking to SpeechCity. This utterance will then create an action in the SpeechCity application which will be passed to the User Model component. This action can then be parsed in order to find out which location the user has requested and then update that preference in the User Model by incrementing its count by 1. This will involve searching the action for the name and identification number of the location being requested and looking that up in the City Model. Any recommendations which are ignored by the user will be determined to have been declined. Since these recommendations are ignored no action will be taken.

7.7 XML Files

The functionality of the User Model component will rely on two XML files. These XML files will be:

1. User Model XML
2. Parser Data XML

Both of these files will play an important role in the running of the system. The User Model XML file is designed to hold the contents of a user’s User Model. This will allow a user to keep their built User...
Model across multiple sessions of SpeechCity. The Parser Data XML file is designed to hold information on the different types of location there are and also keywords that can be looked for in user utterance.

The XML file format will be used for its quick and easy access and update and its ability to store multiple attributes within a single element.

7.7.1 Format of User Model XML
In order to store data from the User Model each entry will be created as an element and each element will contain attributes. These attributes will be:

- **type**: The type attribute will hold the type of a location, such as Café, Bookstore, etc. These types will be taken from the City Model.
- **term**: The term attribute will represent weather the preference is short term or long term. This will only be able to be either “short term” or “long term”.
- **state**: This attribute will describe how a user feels about the preference. This will only ever be “like” or “dislike”.
- **count**: The count attribute will store the amount of times that a use has requested a location of that type. The value must be in the range of -100 to 100.

7.7.2 Format of Parser Data XML
As the system should be able to parse user utterance and update the User Model to reflect the contents of the utterance a list of like and dislike phrases and location types has to be stored. This information will be stored in a separate XML file which will then be used by the FileIO class to populate a lists in the Parser class. The Parser Data XML file will have one element for each keyword and phrase and one element for each keyword which references a location type.

Each like and dislike element will have attributes which specify the keyword and its associated level of likeness. These attributes will be:

- **keyword**: The keyword attribute will hold the word or phrase which will be looked for in the user utterance.
- **weight**: The weight attribute will represent how much a person likes or dislikes something. For example; hate, interested in, adore, detest. The full list can be found in appendix D.

Each of the location type elements will have attributes which store the location type and the keyword that the parse should look for. These attributes will be:

- **keyword**: This attribute will store a word which will be looked for in the user’s utterance.
• **type**: The type attribute will hold the City Model type which is associated with the keyword.

### 7.8 Example User Dialogue

During the design process it is necessary to consider possible ways which the user will interact with SpeechCity and how these interactions will affect the User Model component. The following section describes the possible ways in which a user will interact with the system and the consequences of these interactions. In order to show how these actions affect the User Model component each figure will give an example of the user dialogue with SpeechCity and how the User Model is updated.

For the example in figure 15 the user requests a route to one of their favourite places. SpeechCity is then able to provide the user with a route. This process generates a JSON with information about the requested route, this includes the location name and Id. The User Model component will then parse this JSON to retrieve the type or category from the City Model. The User Model is then updated.

Figure 15 shows and example of a user requesting a route to a location and the affect this has on a User Model that does and does not contain it.
Figure 15: Activity diagram of a user requesting a route to a location.

For the following example in figure 16 the user speaks about one of their preferences, this preference is then picked up by the User Model component and used to update the User Model.
User

“I love Japanese food”

Send to Interaction manager

Send position, utterance and userDA to User Model component

Send utterance to parser

Look for keywords in utterance

Keywords indicate a like

Update User Model

Keywords cannot resolve utterance

Keywords indicate dislike

Update User Model

Discard utterance

(Japanese Food, like, short term, 50)

Figure 16: Activity diagram showing the user utterance and its effect on the User Model
Chapter 8

Implementation

8.1 Overview
This chapter is concerned with the implementation stage of the project. The implementation of the User Model component was divided into three separate stages. The first stage was implementing the design of the software so that a User Model could be built and manipulated by user utterances, user utterances could be parsed, information could be retrieved from the City Model and recommendations could be generated from the City Model information. The second stage was to populate the parsers XML file and the third stage was to integrate the system into the existing SpeechCity application.

8.2 Core functionality
Implementing the core functionality of the User Model component involved building the main system. This was implementing the User Model, the Rules, the parser and the JSON reader.

8.2.1 User Model Implementation
In order to implement the User Model an appropriate data structure had to be chosen. It was decide early on that a Hash Map would be the best structure to represent the preferences of a user. A Hash Map was chosen as it is easily and quickly traversed, every entry has to have a unique key and the structure of key : value seemed to fit nicely with the T : X, Y, Z that had been decided upon in the design stage. This meant that the key of the Hash Map would consist of the City Model type or category and the value would store the attributes of the preference.

The storing of the preference attributes also went through a series of revisions. The initial implementation was a Linked List which would hold each attribute as an element and could then be iterated over to manipulate these attributes. Figure 17 shows the format of the Linked List that had been implemented.

![Example of the LinkedList containing preference attributes.](image)

Due to the amount of attributes, the methods required in order to manipulate each preference and the ease at which the Java programming language can create objects it was decided that each
preference would be better stored as an object. This meant implementing a Preference class, which would hold all information on one preference. The Hash Map would then store the City Model type and the preference object relating to that type. The final Hash Map used to store the user’s preferences is of the type `<String, Preference>`. Figure 18 shows the format of the final Hash Map.

```java
private HashMap<String, Preference> preferences = new HashMap<String, Preference>();
```

Figure 18: Example of the Hash Map used to represent the user’s preferences.

### 8.2.2 Preferences

The initial implementation of the preferences included a weighting attribute. This weighting could be either Low, Medium or High. Figure 19 shows how this implementation was formatted.

```xml
<Preference count="0" state="like" term="short term" weight="Low" type="cafe"/>
<Preference count="0" state="like" term="short term" weight="Medium" type="shop"/>
<Preference count="0" state="like" term="short term" weight="High" type="restaurant"/>
```

Figure 19: Example of original preference attributes.

However, after discussions about this implementation approach it was decided that this attribute could not accurately depict a user’s liking of a preference. This approach raised the question “What was the difference between a preference that has a weighting of High and one that has a weighting of Low?” To solve this problem we decided that the weight should be removed and the count be used.

As stated in chapter 7 section 7.3.1 the remaining attributes are `term` and `state`, implemented into the system as was planned in the design stage.

### 8.2.3 Rules

During the design stage it was decided that the Rules would be split into two groups. Each rule falls into one of two groups; recommendation rules or update rules.

#### 8.2.3.1 Recommendation Rules

Each attribute of a preference is used in order to assign it a score. This score is based on what each attribute’s value is and is initially set to zero. To generate the overall score we first look at the `state` attribute. The preference score is updated based on whether it is like or dislike. This is done using the `stateScore` method, as seen in figure 20.
private static double stateScore(String type, UserModel model) {
    double score = 0.0;
    if (model.getPreferenceAt(type).getState().equals("like"))
        score = 1.0;
    else
        score = -100.0;
    return score;
}

Figure 20: stateScore method.

Once this has been applied the **termScore** method is used. This updates the score based on whether the preference is a long term or short term preference, this can be seen in figure 21.

```java
private static double termScore(String type, UserModel model) {
    double score = 0.0;
    if (model.getPreferenceAt(type).getState().equals("like")) {
        if (model.getPreferenceAt(type).getTerm().equals("short term"))
            score = 1.0;
        else
            score = 2.5;
    } else {
        if (model.getPreferenceAt(type).getTerm().equals("short term"))
            score = -50.0;
        else
            score = -100.0;
    }
    return score;
}
```

Figure 21: termScore method.

In order to deal with the count score we first have to order all preferences by their count value into a list. This is done so that the first entry in the ordered list of preferences is the one with the highest count value. From here we assign a score based on their position in the list. To do this we create a decrement value which will be used to change the score as we go down the list; maxScore = 100 and count = number of types in the list, figure 22 shows this.

```
decrement = (int) (maxScore / count);
```

Figure 22: Decrement formula.

Once this has been calculated we can then assign the score to each type, as seen in figure 23.

```
maxScore = maxScore - decrement;
score += maxScore;
```

Figure 23: Assigning score based on decrement.
Once each preference has been assigned a score they are added to another ordered Hash Map, in the order of highest score first. This is so the recommendation calculations know what the highest liked preferences are to recommend. This can be seen in figure 24.

\[
\text{Map<String, Double> orderedMap = new LinkedHashMap<String, Double>();}
\text{orderedMap = sortByValue(unorderedMap);}\]

Figure 24: Creation of the ordered map.

In order to recommend the closest locations to the user the methods `crowFliesDistance` and `getClosestEntity` are used. The `crowFliesDistance` calculates the distance from the user to the location to be recommended, this is used by the `getClosestEntity` method. The `getClosestEntity` method puts each location through the `crowFliesDistance` method and returns the location which is closest to the user.

### 8.2.3.2 Update Rules

These rules are used to ensure that the `state` and `term` attributes are the correct value for the value that is contained in the `count` attribute. Two methods are used:

1. **updateState**: Ensures that the state is like if `count > 0` and dislike if `count < 0`.
2. **updateTerm**: Ensures that the term is “long term” if `count > 75` for like and `< -75` for dislike and “short term” if `count < 75` for like and `> -75` for dislike.

### 8.2.4 City Model Data

In order to generate a set of recommendations for a user the data has it be retrieved from the City Model. This functionality is implemented using the JSONReader class. This class provides methods which build the URL to the City Model, retrieve the Data, and return the data in the form of a JSON. This allows the data to be directly sent to the Interaction Manager without any further formatting. Figure 25 presents the `buildURL` method which builds the URL to the City Model.

```java
private String buildURL(String location, String distance, String type) {
    String url = "http://23.227.172.76:8001/citymodel/getNearbyEntitiesInfoByType/"
    + location + "/" + distance + "/" + type + ":";
    return url;
}
```

Figure 25: buildURL method to get URL to City Model.

Figure 26 shows the `readAll` method. This method retrieves the data from the City Model.
Finally the `readJson` method is used. This method uses both of the above methods in order to format the retrieved data into a JSON which can be read by the Interaction Manager and can be seen in figure 27.

```java
private JSONObject readJson(String location, String distance, String type) throws IOException, JSONException {
    InputStream in = new URL(buildURL(location, distance, type)).openStream();
    BufferedReader br = new BufferedReader(new InputStreamReader(in, Charset.forName("UTF-8")));
    String jsonText = readAll(br);
    JSONObject json = new JSONObject(jsonText);
    return json;
}
```

Figure 27: readAll method used to format the City Model data into a JSON.

### 8.2.5 Calculating Recommendations

The main function of the system is to recommend locations to the user. To implement this functionality the method `runCalculations` was created. Using Java’s `ScheduledExecutorService` this method executed the recommendation calculations every thirty seconds. This was achieved by using the following lines of code:

```java
ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);
executor.scheduleAtFixedRate(runCalculations, 0, 30, TimeUnit.SECONDS);
```

### 8.2.5.1 Generation of Recommendations

The generation of the recommendations to be given to the user was implemented using the Rules class. In order to offer the most suitable locations a list of the users preferences is compiled, which stores each preference along with the score it has received by applying the recommendation rules to it. The populated list is stored in a Linked Hash Map in order to preserve the ordering. The `run` method then makes calls to the City Model using each element in the Linked Hash Map until it receives a JSON.
with entities, any JSON that is empty is discarded as this means there are no locations within 2000 meters of the user. Figure 28 shows the code that is used to generate the recommendations.

```
    orderedTypes = Rules.getTypeOrder(model, h);

    for (Entry<String, Double> entry : orderedTypes.entrySet()) {
        JSONArray locations = new JSONArray();
        if (!pastRecommendations.contains(entry.getKey())
            && entry.getValue() >= 0) {
            try {
                // Use the coordinates to get the details of a location
                // based on the order from the User Model
                if (entry.getKey().equals("cafe")
                    || entry.getKey().equals("shop"))
                    locations = jsonReaderjsonp(coordinates,
                        shortDISTANCE,
                        entry.getKey().replace(" ", "%20");
                else
                    locations = jsonReaderjsonp(coordinates,
                        DISTANCE, entry.getKey()).replace(" ", "%20");
            } catch (FileNotFoundException f) {
                f.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

Figure 28: Code showing how the recommendations are generated.

8.2.6 Parser

Another main function of the User Model component is to listen to any user utterances and update the User Model if anything is said, from which conclusions can be made about their preferences. In order to do this the system has to implement some kind of parser. As described in chapter 7 section 7.7.2 and section 8.3 a XML file is used to store keywords and phrases along with their respective City Model types.

The initial implementation had the parser running every five seconds checking for any utterances spoken by the user. This was achieved in the same way as the recommendation calculations, using the `ScheduledExecutorService`.

8.2.6.1 Parsing User Utterance

The Parser class contains the method `parseNonAnswer`. This method deals with parsing any utterances which are not related to the requesting of routes. This method has been implemented to search for keywords in the user utterance and try to match them to the keywords stored in the XML
file. In order to do this two data structures are needed. Two Hash Maps are created which hold the data stored in the XML file. Figure 29 shows the creation of these two Hash Maps.

```java
private Map<String, String> stateKeywords = new HashMap<String, String>();
private Map<String, String> types = new HashMap<String, String>();
```

Figure 29: Creation of Hash Maps which store parser data.

The `parseNonAnswer` method iterates over both of these maps checking if the utterance contains anything from both lists. First the utterance is checked for any words in the `stateKeywords` map. This looks for common words and phrases which express a like or dislike. If a match is found then the `types` map is iterated over to find what location type the user is speaking about. Figure 30 shows how this code is implemented in the `parseNonAnswer` method.

```java
public void parseNonAnswer(String utterance, UserModel model) {
    for (Entry<String, String> keyword : types.entrySet()) {
        for (Entry<String, String> entry : stateKeywords.entrySet()) {
            if (!entry.getKey().contains("not") & !entry.getKey().contains("don't")) {
```

Figure 30: parseNonAnswer method iterations to match keywords in user utterance.

If a match is found between the utterance and the `type` map then the User Model can be updated. In order to best reflect the implications of the keyword used in the utterance the `stateKeywords` map uses its values (see in chapter 8 section 8.3) to determine how much a user likes the type in question. Checks are made to find out what level of interest the user has in the type they have spoken about and the User Model is updated as shown in figure 31.
8.3 Parser XML file

In order to collect information about a user implicitly we had to make sure that any utterances connected to the preferences of a user could be parsed and used to update their User Model (this can be seen in greater detail in chapter 7 section 7.6). This involved the creation of an XML file that is used by the parser class and is populated with common phrases and words and the different types of locations to look out for. The structure of the XML file contains the following elements:

- **parserData**: The root element.
- **types**: Contains the pref elements.
- **states**: Contains the states elements.
- **pref**: hold data about the different locations that the system looks out for.
- **state**: Holds data about common words and phrases and their associated weight.

To start, the most common words and phrases that express a user’s like or dislike of something were added to the XML file. These were words and phrases such as; love, hate, adore, favourite food, no interested, etc. Each word and phrase added to the XML file has an associated weight level which tells the parser the level of like or dislike. These weights are used by the parser class in order to best
estimate how much a user likes or dislikes something. The weights that can be associated with a word or phrase are:

- **High**: This can be either likeHigh or dislikeHigh.
- **Medium**: This can be either likeMedium or dislikeMedium.
- **Low**: This can be either likeLow or dislikeLow.

These words, phrases and associated weight were added to the XML file under the `<states>` element. Each word, phrase and weight is then added to the file under its own `<state>` element setting attributes for each `<State>` element to hold the data. Figure 32 gives an example of some `<state>` elements within the XML file, showing the attributes for each element.

```xml
<state keyword="savour" weight="likeHigh" />
<state keyword="interested" weight="likeMedium" />
<state keyword="interest in" weight="likeMedium" />
<state keyword="do not like" weight="dislikeLow" />
<state keyword="don't like" weight="dislikeLow" />
<state keyword="Hate" weight="dislikeHigh" />
```

*Figure 32: Examples of the `<state>` elements in the parserData.xml file.*

The other part of the XML file stores the types which are used to update the User Model. Each type is stored with the keyword(s) that the parser looks for in any user utterances. These keywords and types are stored in the attributes of the `<pref>` elements, which in turn added to the `<types>` element. Figure 33 gives an example of some of the `<pref>` elements in the XML file, showing the attributes for each element.

```xml
<pref keyword="art" type="Art Gallery" />
<pref keyword="art gallery" type="Art Gallery" />
<pref keyword="art galleries" type="Art Gallery" />
<pref keyword="museum" type="Museum" />
<pref keyword="museums" type="Museum" />
<pref keyword="jewellery" type="Jewelry Store" />
<pref keyword="sporting goods" type="Sporting Goods Shop" />
<pref keyword="spa" type="Spa" />
<pref keyword="cosmetics" type="Cosmetics Shop" />
<pref keyword="make-up" type="Cosmetics Shop" />
```

*Figure 33: Examples of `<pref>` elements in the parserData.xml file.*

The complete parser XML file used by the User Model component can be found in appendix D.
8.4 Integration with SpeechCity

In order for the User Model component to function completely the code had to be integrated into the existing SpeechCity system. The User Model component required the user’s position, in order to calculate the nearest locations to recommend, and any user utterances. The integration with SpeechCity also allowed the recommendations to be sent to SpeechCity’s Interaction Manager and then presented to the user.

The integration process involved technical meetings with the developers of SpeechCity to decide the most efficient way to send and receive data to and from the component and the Interaction Manager. As stated in chapter 7 the User Model component was implemented to make calls to the SpeechCity system in order to get the user position, utterance and send the recommendations. However, it was decided that it would be more efficient and practical for the Interaction Manager to make calls to the User Model component, rather than receiving calls.

8.4.1 Tailoring core functionality to SpeechCity requirements

This change in the way the User Model component was to send recommendations and receive the information needed to perform the recommendation calculations meant that the implementation had to be changed.

Rather than the User Model component performing calculations every thirty seconds and looking for and parsing user utterances every five seconds the Interaction manager calls both parts of the component every two seconds, this meant that the component code no longer needed to have an event scheduler, and the recommendations would be presented to the user every two minutes. Calculating recommendations every two seconds ensures that the most up to date user position is used when the recommendations are finally presented to the user, giving recommendations closest to the user.

The following two subsections discuss the changes made during the integration of the recommendation calculations and the parser into SpeechCity.

8.4.1.1 Integrating the recommendation calculations

The recommendation calculations were implemented to run every thirty seconds. This was done under the assumption that the User Model component would be making requests to SpeechCity’s Interaction Manager for the required data. This had to be changed due to the Interaction manager now calling the User Model component, this involved removing the code relating to the event scheduler thread.
It was decided during the technical meetings that the most efficient way of sending the user’s position and utterance to the User Model component was through the use of a JSON. The JSON is passed by the Interaction Manager to the User Model component each time the Interaction Manager makes a call to it, Figure 34 shows the format of the input JSON and gives example data for each attribute.

```json
{
"userPosition": "55.9459916117002, -3.18971365502437",
"userUtterance": "I have gone off beer"
}
```

Figure 34: Example of coordinates and utterance sent to User Model component.

This JSON can then be parsed to obtain the user’s position and any utterance, which will be null if nothing has been said to the system.

In order to send the recommendations back to the Interaction Manager and be presented to the user a new output JSON is created. During the calculation of the recommendations the most appropriate location’s JSON data, from the City Model, is added to the output JSON. Once all recommendations have been calculated a JSON is sent back to the Interaction manager consisting of both recommendations. Figure 35 shows the format of the output JSON used by the Interaction manager to receive the recommendations, giving example data for each attribute of both location’s JSONs.

```json
{
"recommendations": [
{
"OSM type": "node",
"name": "Olly Bongo's",
"id": "366191100",
"position": "-3.1898846 55.9457304",
"type": "cafe",
"Foursquare rating": {
"address": "4+Teviot+Pl",
"city": "Edinburgh",
"FoursquareValidity": "3",
"postcode": "EH1+2QR",
"telephone": "+44131+225+2849",
"checkins": "239",
"url": ""
}
}
],
```

Figure 35: Example of the format of the output JSON sent to the Interaction Manager.

### 8.4.1.2 Integrating the parser

The implementation of the parser side of the User Model component had to be altered in order to allow SpeechCity’s Interaction Manager to make calls to it. As shown in the above section the
Interaction Manager passes the user’s utterances as part of the input JSON. This allows the utterance to be parsed and stored in a new String, which is then able to be used by the parser class (described in Chapter 7 section 7.2.2).

In order to update a User Model based on where a user travels the User Model component is sent a JSON which contains the userDA (user Dialogue Act) from the Interaction Manager.

The userDA contains the current tasks of what the Interaction Manager has to do, for example this might be requesting a route. Figure 36 shows an example of the format of a userDA JSON used by the Interaction Manager.

```
"userDA": {
  "cf": "requestRoute",
  "parserConfidence": 100,
  "dimension": "task",
  "entity": {
    "entityName": "costa",
    "entityDuplicates": "true",
    "entityId": "2111165932",
    "entityRefType": "cmName"
  }
}
```

![Figure 36: Example of the userDA JSON.](image)

This userDA JSON is passed to the User Model component in the same input JSON as the user’s position and utterances. Figure 37 gives an example of the complete input JSON which is passed to the User Model component from the Interaction Manager.

```
{
  "userDA": {
    "cf": "requestRoute",
    "parserConfidence": 100,
    "dimension": "task",
    "entity": {
      "entityName": "costa",
      "entityDuplicates": "true",
      "entityId": "2111165932",
      "entityRefType": "cmName"
    }
  },
  "userPosition": "55.9459916117002, -3.18971365502437",
  "userUtterance": "i have gone off beer"
}
```

![Figure 37: Complete input JSON received by the User Model component](image)
The run method had to be changed to make checks to find out if there is a userDA to parse and if there is call the relevant `parseUserDA` method. The `parseUserDA` method looks for the “request route” task, this indicates that the user has made a request to SpeechCity for directions to a location, and updates the User Model based on the information contained within the userDA JSON. The type and the Identification number of the requested location are retrieved from the userDA JSON and are used to look up the information on that Identification number in the City Model. This is done by using the `buildURL` method in the JsonReader class. Figure 38 shows an example of the `buildURL` method which constructs the URL needed to make the call to the City Model to look up the information of a location given an Identification number.

```java
private String buildURL(String Id) {
   String url = "http://23.227.172.76:8001/citymodel/getKnrityInfoById/" + Id + ";
   return url;
}
```

Figure 38: `buildURL` method used to retrieve JSON from City Model using Id.

This call to the City Model is carried out as it allows us to retrieve the category of the requested location and either update or add that category in the User Model, the category is used instead of the type to provide a more varied User Model (this is described in greater detail in chapter 7 section 7.5).

### 8.4.2 User Model Component Integrated into SpeechCity

The following screenshots show how the User Model component functionality is implemented into the SpeechCity Android application.

Figure 39 shows how the recommendations are presented to the user.
Figure 39: Recommendations presented to the user in SpeechCity.

The following figure shows how the recommendations are shown to the user on the map interface.

Figure 40: Example of the recommendations displayed on the map in SpeechCity.
Chapter 9

Evaluation

9.1 Overview
This chapter aims to evaluate the success of the User Model component integrated into SpeechCity. The component will be evaluated to ensure that all requirements gave been successfully implemented and to discover how successful the component has been. Furthermore, the evaluation strategy is presented along with the types of tasks and users that participated in the evaluation. A critical analysis of the results of the evaluation questionnaire is presented along with the conclusions on how successful the end system is. The aim of the evaluation is to test a system with a populated User Model vs a system without a populated User Model.

9.2 Evaluation Strategy
Before any participant undertook the evaluation a user testing informed consent form had to be signed, to ensure that permission was obtained to use any data collected during the evaluation sessions. An example of the user testing informed consent form that was given out to each of the participants can be found in appendix A.

After discussions with the project supervisor and the SpeechCity development PhD student it was decided that the goal of the evaluation was to find out how successful the User Model component is. This involved finding out if the component had achieved its objectives, how well it was able to perform and how useful the recommendations were to the participants.

The strategy was to distribute the SpeechCity application to each of the participants and have them explore Edinburgh while using the system. The participants were to spend as much time as possible using the system. This included finding routes to their favourite locations and speaking about their preferences. The data collected during each session is then used in order to reach a conclusion and answer the project hypothesis of “a system with a populated User Model is more desirable than a system without a populated User Model”.

9.3 Evaluation tasks and Survey
A series of tasks were conceived in order to best evaluate the performance and usefulness of the User Model component. The tasks were simple instructions which the participant would follow which
would cause the User Model to be updated with some preferences of the participant. An example of the task sheet that was given to each participant at the start of the evaluation session can be found in appendix B.

An evaluation session both started and ended with each participant completing a survey. One was to be completed before the use of the User Model component and the other to be completed after the User Model component had been used and the tasks had been completed. The results of both surveys could then be compared, showing if the system did in fact operate as it should and its inclusion was helpful and desirable.

The survey starts with a question asking the participant to rate the recommendations on a scale of 1 to 10. The survey then consists of a series of statements with the following six point Likert scale of agreement: strongly disagree -> disagree -> slightly disagree -> slightly agree -> agree -> strongly agree. The participants were to circle which best matched their agreement to the statement. A section at the end of the survey allowed any comments to be written.

An example of both surveys can be found in appendix C.

9.4 Evaluation Participants
The evaluation did not require any expertise knowledge or training in order to use the SpeechCity Android application. However, SpeechCity requires the Android operating system therefore, participants with access to a device running Android OS 4.2 and above were preferred. As there are many different devices with different operating systems available any participant that did not meet the required Android operating system was given a device to test the system with.

A mixture of genders and age groups was sought and all participants were given the same instructions and tasks to complete. There was no separation of the participants based on any factors, such as ability, prior knowledge, etc., as it was felt that this best reflected the extensively varied target audience of the SpeechCity application.

9.5 Analysis of questionnaire results
As stated above the evaluation consisted of participants completing tasks and two surveys. This results analysis intends to show the differences between the pre-use and post-use surveys and use this to conclude how successful the User Model component has been.

There were ten participants in total evaluating the system and on average each participant used the system for one hour and thirty minutes.
In order to best compare the two surveys this section looks at each question from both surveys in turn, analysing the differences between the two and discussing what can be deduced from said differences. The Wilcoxon Signed-Rank test has been performed on questions 1, 2, 3, 4, 8, 9 and 10 in order to determine if there is a significant difference between the Pre-Use and Post-Use surveys. This will be a one-tailed test as we expect to find that the populated User Model is better than the default User Model. The Wilcoxon Singed-Rank test was not performed on questions 5, 6 and 7 because the results were either the same or similar with no real significant difference between the Pre-Use and Post-Use surveys.

Question 1 presents the numbers in a scale of 1 to 10 and questions 2 to 10 use the six point Likert scale shown in table 7.

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strongly disagree</td>
<td>Disagree</td>
<td>Slightly disagree</td>
<td>Slightly agree</td>
<td>Agree</td>
<td>Strongly agree</td>
</tr>
</tbody>
</table>

The default User Model used in the Pre-Use Survey for each participant can be seen below in figure 41.

```xml
<UserModel>
    <Preferences>
        <Preference count="0" state="like" term="short term" type="cafe"/>
        <Preference count="0" state="like" term="short term" type="shop"/>
        <Preference count="0" state="like" term="short term" type="named building"/>
        <Preference count="0" state="like" term="short term" type="restaurant"/>
    </Preferences>
</UserModel>
```

**Figure 41: Default User Model.**

9.5.1 Question 1

Question 1 asked the participant to rate, on a scale of 1 to 10, how relevant the recommendations were to them. During each evaluation session participants were asked to walk around the city until they had received ten recommendations and were then to complete the pre-use survey. As stated earlier the post-use survey was completed after all tasks had been completed. The average results for question 1 are shown in figure 42.
The results from question 1 show that the default User Model was not suitable for most users. After using the system and populating the User Model the recommendations were vastly improved and the majority of participants agreed, this is seen in the Post-Use average of 8.4.

Applying the Wilcoxon Signed-Rank test produced the following:

\[ W\text{-Value: 0} \]
\[ p\text{-Value: 0.00256} \]
\[ \text{Critical-Value: 10} \]

As the W-value is < the Critical-Value and the p-Value < 0.05 the Wilcoxon test showed that the use of a populated User Model significantly improved the recommendations received by the participant (W(0) = 10, p < 0.05, one- tailed test).

**9.5.2 Question 2**

Question 2 posed the statement “The system recommended things I was interested in” and the participant was to choose how strongly they agreed or disagreed with the statement. The average results for question 2 are shown in figure 43.
The results from question 2 show that after the use of the system the participant starts to receive recommendations for locations that they were interested in, with an average score of “agree”. Compare this to the Pre-Use average and we can see that this is proven.

Applying the Wilcoxon Signed-Rank test produced the following:

\[
\begin{align*}
W\text{-Value: } & 0 \\
p\text{-Value: } & 0.00256 \\
\text{Critical-Value: } & 10
\end{align*}
\]

As the W-value is < the Critical-Value and the p-Value < 0.05 the Wilcoxon test showed that after a User Model has been populated the participant started to receive recommendations they were interested in (W(0) = 10, p < 0.05, one-tailed test).

9.5.3 Question 3

Question 3 posed the statement “The system changed its recommendations based on where I went”. The average results for question 3 are shown in figure 44.
Here the results show that even when the default User Model is being used there is still a good amount of locations being recommended.

Applying the Wilcoxon Signed-Rank test produced the following:

\[ W-Value: 12 \]
\[ Critical-Value: 3 \]

As the W-value is > the Critical-Value the Wilcoxon test showed that there is no significant difference between the verity of recommendations given by the default and a populated user model \((W(12) = 3, p > 0.05, \text{one-tailed test})\).

**9.5.4 Question 4**

Question 4 posed the statement “The system seemed to know my preferences”. The average results for question 4 are shown in figure 45.

![Figure 45: Average results from question 4.](image)

The results for question four further justify the User Model Components success. The Pre-Use average shows that the system did not know the participants’ preferences while an average of “agree” suggests that most participants recognised that the User Model was recommending their preferences.

Applying the Wilcoxon Signed-Rank test produced the following:

\[ W-Value: 0 \]
\[ p-Value: 0.00256 \]
\[ Critical-Value: 10 \]

As the W-value is < the Critical-Value and the p-Value < 0.05 the Wilcoxon test showed that after use of the system participants found that it knew some of their preferences \((W(0) = 10, p < 0.05, \text{one-tailed test})\).
9.5.5 Question 5

Question 5 posed the statement “I found it easy to access the recommendations”. The average results for question 5 are shown in figure 46.

![Question 5](image)

Figure 46: Average results from question 5.

Here we can see that both the Pre-Use and Post-Use surveys slightly agree that the recommendations are able to be viewed with ease.

9.5.6 Question 6

Question 6 posed the statement “The frequency of recommendations was correct”. The average results for question 6 are shown in figure 47.

![Question 6](image)

Figure 47: Average results from question 6.

The results for question 6 confirm that the frequency at which the recommendations are presented to the user is good and did not become annoying.

9.5.7 Question 7

Question 7 posed the statement “I was able to find all the information that I needed”. The average results for question 7 are shown in figure 48.
These results show that the information was easy to access and each participant was able to find the information they required. The results also show that the average score was slightly higher for the Post-Use survey implying that the User Model was able to provide more information.

9.5.8 Question 8

Question 8 posed the statement “The recommendations received were appropriate for me. The average results for question 8 are shown in figure 49.

These results confirm that the User Model is able to learn a user’s preferences. The average score of “agree” shows that the majority of the participants thought that the User Model component had learned their preferences, this is further shown by the “disagree” score received before Use of the system.

Applying the Wilcoxon Signed-Rank test produced the following:

W-Value: 0
p-Value: 0.00256
Critical-Value: 10

As the W-value is < the Critical-Value and the p-Value < 0.05 the Wilcoxon test showed that with a populated User Model the system recommended locations suitable for each participant (W(0) = 10, p < 0.05, one-tailed test).

9.5.9 Question 9

Question 9 posed the statement “The locations recommended were varied”. The average results for question 9 are shown in figure 50.

![Figure 50: Average results from question 9.](image)

Here we can see that the default User Model does provide a varied series of recommendations. However, a populated User Model does provide a greater variance. This is shown by the “strongly agree” for the Post-Use and “agree” for the Pre-Use.

Applying the Wilcoxon Signed-Rank test produced the following:

**W-Value: 6**

**Critical-Value: 3**

As the W-value is > the Critical-Value the Wilcoxon test showed there is no significant difference between the default and a populated User Model in the verity of its recommendations (W(6) = 3, p > 0.05, one-tailed test).

9.5.10 Question 10

Question 10 posed the statement “I would use this system in the future”. The average results for question 10 are shown in figure 51.
Finally from question 10’s results we can see that after the Participant has populated their User Model the system is more desirable. The average score of “agree” shows that the system provided recommendations that were useful and accurate.

Applying the Wilcoxon Signed-Rank test produced the following:

\[
\begin{align*}
W{-}\text{Value}: & \quad 0 \\
p{-}\text{Value}: & \quad 0.00256 \\
\text{Critical{-}Value}: & \quad 10
\end{align*}
\]

As the \( W{-}\text{value} \) is < the Critical{-}Value and the \( p{-}\text{Value} \) < 0.05 the Wilcoxon test showed the system with the populated User Model was significantly more likely to be used in the future (\( W(0) = 10, p < 0.05 \), one{-}tailed test).

### 9.6 Conclusion of Evaluation

By looking at the results from section 9.5 we can make an overall conclusion of how successful the User Model component is, and in turn how successful the project has been. Furthermore, we can take a look at how successful the approach to the evaluation was.

The results as a whole show that the User Model component is able to learn what a user likes and recommend locations which a user should like. The Wilcoxon Signed-Rank tests shows that there is a significant difference between the recommendations made between the default User Model and a populated User Model, the populated User Model is able to recommend locations suitable for each user. The results show that a default User Model was still able to recommend a good variety of locations even though the majority of locations were not appropriate for the user. As each participant used the system for different lengths of time it can also be seen that the longer a user used the system, requesting routes and speaking about their preferences, the better the recommendations were. Using the system for a longer time almost eliminated the default entries in the User Model, so we can
conclude using the system over an extended period of time, which in effect would produce a vastly populated User Model, will provide the user useful and desirable recommendations every time.

The evaluation approach was successful. The only variable that was not under control was the length of time which a participant used the system.
Chapter 10

Summary

10.1 Overview
This chapter aims to summarise the achievements of the project as a whole as well as present the limitations of the User Model component. Furthermore, a discussion on the possible extensions and future work that could be undertaken is presented. Finally the conclusions reached are stated and explained.

10.2 What Has Been Achieved
The User Model component integrated into the SpeechCity application has been demonstrated to have successfully achieved all of the objectives set out in chapter 1 section 1.2.2 and has implemented all of the functional and non-functional requirements from chapter 4 section 4.4. The following list shows that each objective has been achieved.

- **Define the User Model**: User Model has been defined and implemented.
- **Collect user information**: Information on user is collected implicitly.
- **Build the User Model**: User Model is built using information collected.
- **Find nearby Locations to recommend using the User Model**: Nearby locations are retrieved for the City Model
- **Relay recommendations to the user**: user is presented with two recommendations every two minutes.
- **Distinguish long term from short term preferences**: Preference can be either a long or short term preference.
- **Update the User Model**: Update rules handle updating the User Model.
- **Integrate the User Model Component into the SpeechCity Android application**: Integration was successful.
- **Evaluate the success of User Model**: Evaluation has been done.

10.3 Limitations and Future Work
There are several aspects to the User Model component which limit its potential and that can be improved upon in the future. The main limitation is the way that it retrieves the nearby locations from the City Model. Querying the City Model takes time, the greater the distance from the user the more...
locations there are and hence the longer the query will take. The User Model component will look for locations based on the users highest rated preferences, this means that what is recommended might be a great deal further away than a preference which is almost rated the same.

With more time to continue this project the parserData.xml file would be populated to include more common words and phrases and more City Model types and their respective locations. This would enhance the parser functionality of the system.

A distance check could be introduced. This could be used in order to suggest locations which may not have the highest score but are closer to the user. For example, a preference with a score of 90 may be recommended over a preference with a score of 100 if the former is significantly closer to the user.

Finally the difference between long term and short term preferences would be expanded upon. This would be developed to include the passage of time. So that short term preferences would change to long term preferences based on time rather than their count. This did not seem feasible for this project as the evaluation was not conducted over a long period of time.

Below is a summarised list of the possible future work that could be done.

- Improve efficiency of City Model calls.
- Expansion of parserData.xml file.
- Account for distance between preferences.
- Use time to differentiate long term and short term preferences.

10.4 Conclusions

This project aimed to design and implement a new component to be integrated into the SpeechCity Android application. This new component was to collected information implicitly on a user and construct User Model, which can then be used to recommend locations which a user will like. Moreover, the component also listens for utterances about the preferences of a user and updates the model accordingly. A review of the current literature is presented looking at User Modelling and how it has been used, Spoken dialogue systems and data collection.

In order to satisfy the aim of the project the User Model component has been developed and integrated into the SpeechCity application. The system collects data based on requested routes and the utterances of a user and builds a User Model and recommends locations based on a series of rules to determine the most suitable, the longer a user uses the system the more complete the model will be.
An evaluation of the success of the component has been performed and presented. This was designed to contrast the results from using a default User Model and one that had been populated by a user using SpeechCity to explore a city. The results showed that the system was able to successfully learn a user’s preferences and recommend suitable locations based on those preferences which the user found interesting and that the majority would use the system again.
References


Appendix A: User Testing Informed Consent Form

USER TESTING INFORMED CONSENT FORM

<table>
<thead>
<tr>
<th>Study Administrator:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant Name:</td>
</tr>
</tbody>
</table>

The goal of this study is to evaluate user experience of the SpeechCity Android Application. Your participation will help us achieve this goal.

In this session you will be working with an Alpha version of SpeechCity. We’ll ask you to perform tasks a typical user might do, such as exploring the city, requesting routes to your favourite places and viewing recommended locations.

All information collected in the session belongs to Heriot Watt University and will be used for research purposes. We may publish our results from this and other sessions in our reports, but all such reports will be confidential and anonymous: they will not include your name.

This is a test of the software. We are not testing you. You may take breaks as needed and stop your participation in the study at any time.

STATEMENT OF INFORMED CONSENT

I have read the description of the study and of my rights as a participant. I voluntarily agree to participate in the study.

Print Name: ____________________________________________________________
Signature: ____________________________________________________________
Date: __________________________________________________________________
Appendix B: Evaluation Tasks

SpeechCity Evaluation Tasks

The purpose of these tasks are to evaluate user experience of the SpeechCity App. This is done by completing a series of tasks using SpeechCity.

Recommended locations can be found by clicking the ‘dialogue’ tab of the SpeechCity interface. You will be informed by the App when new recommendations are available.

Please do the following tasks. If the app stops working, please restart it.

1. Start the application and wait to be notified of available recommendations.
2. View the recommendations made by the App.
3. Wait to be notified of new recommendations and once received view them. Do not speak to the app during this.
4. Please do step 3 until you have been presented with at least 10 sets of recommendations. You should walk around the city while waiting.
5. You should now complete the Pre-Use survey.
6. Request a route to your favourite restaurant (e.g. “Directions to The Outsider”)
7. Request a route to a named location of interest to you (this can be a museum, library, sports hall, etc.).
8. Request a route to your favourite shop (for example, Waterstones, Game, etc.).
9. Wait for some more recommendations. Look at the locations that have been recommended to you and if you like one request a route to it.
10. Please continue to explore the city using SpeechCity, repeating tasks 6 to 9 for as long as possible.
11. Using the recommendations try to find a restaurant of your favourite type (e.g. Indian)
12. Using the recommendations try to find a location of a type that interests you (e.g. art gallery).
13. Using the recommendations find a location that you like but have never been to.
14. You should now fill out the Post-Use Survey.

Evaluation is complete.

Thank you for your time and feedback.
Appendix C: Pre-Use and Post-Use Survey

SPEECHCITY QUALITY SURVEY

PRE-USE SURVEY

Name: __________________________ Age: __________________________

1. On the following scale of 1 to 10, where 10 is best, please circle the number which represents how appropriate the recommendations were:

   1  2  3  4  5  6  7  8  9  10

   For each item identified below, circle the number to the right that best fits your judgment of its quality. Use the rating scale to select the quality number.

<table>
<thead>
<tr>
<th>Survey Item</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. The system recommended things I was interested in.</td>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>3. The System changed its recommendations based on where I went.</td>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>4. The system seemed to know my preferences.</td>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>5. I found it easy to access the recommendations.</td>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>6. The frequency of recommendations was correct.</td>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>7. I was able to find all the information that I needed.</td>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>8. The recommendations received were appropriate for me.</td>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>9. The locations recommended were varied.</td>
<td>1  2  3  4  5  6</td>
</tr>
<tr>
<td>10. I would use this system in the future.</td>
<td>1  2  3  4  5  6</td>
</tr>
</tbody>
</table>

Please write any comments regarding the system in the box below.
### SPEECHCITY QUALITY SURVEY

#### POST-USE SURVEY

**Name:**

**Age:**

1. On the following scale of 1 to 10, where 10 is best, please circle the number which represents how appropriate the recommendations were:

   1   2   3   4   5   6   7   8   9   10

   
   For each item identified below, circle the number to the right that best fits your judgment of its quality. Use the rating scale to select the quality number.

<table>
<thead>
<tr>
<th>Survey Item</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.   The system recommended things I was interested in.</td>
<td>Strongly Disagree</td>
</tr>
<tr>
<td>2.   The system recommended things I was interested in.</td>
<td>1</td>
</tr>
<tr>
<td>3.   The System changed its recommendations based on where I went.</td>
<td>1</td>
</tr>
<tr>
<td>4.   The system seemed to know my preferences.</td>
<td>1</td>
</tr>
<tr>
<td>5.   I found it easy to access the recommendations.</td>
<td>1</td>
</tr>
<tr>
<td>6.   The frequency of recommendations was correct.</td>
<td>1</td>
</tr>
<tr>
<td>7.   I was able to find all the information that I needed.</td>
<td>1</td>
</tr>
<tr>
<td>8.   The recommendations received were appropriate for me.</td>
<td>1</td>
</tr>
<tr>
<td>9.   The locations recommended were varied.</td>
<td>1</td>
</tr>
<tr>
<td>10.  I would use this system in the future.</td>
<td>1</td>
</tr>
</tbody>
</table>

Please write any comments regarding the System in the box below.
Appendix D: Parser Data XML File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<parserData>
  <types>
    <pref keyword="italian" type="Italian Restaurant" />
    <pref keyword="indian" type="Indian Restaurant" />
    <pref keyword="thai" type="Thai Restaurant" />
    <pref keyword="chinese" type="Chinese Restaurant" />
    <pref keyword="japanese" type="Japanese Restaurant" />
    <pref keyword="sandwich" type="Sandwich Place" />
    <pref keyword="sandwiches" type="Sandwich Place" />
    <pref keyword="chip shop" type="Fish & Chips Shop" />
    <pref keyword="chippy" type="Fish & Chips Shop" />
    <pref keyword="chips" type="Fish & Chips Shop" />
    <pref keyword="post office" type="Post Office" />
    <pref keyword="coffee" type="Coffee Shop" />
    <pref keyword="african" type="African Restaurant" />
    <pref keyword="fast food" type="Fast Food Restaurant" />
    <pref keyword="bakery" type="Bakery" />
    <pref keyword="cupcake" type="Cupcake Shop" />
    <pref keyword="clothes" type="Clothing Store" />
    <pref keyword="clothing" type="Clothing Store" />
    <pref keyword="turkish" type="Turkish Restaurant" />
    <pref keyword="burrito" type="Burrito Place" />
    <pref keyword="burritos" type="Burrito Place" />
    <pref keyword="fried chicken" type="Fried Chicken Joint" />
    <pref keyword="seafood" type="Seafood Restaurant" />
    <pref keyword="music" type="Music Venue" />
    <pref keyword="pizza" type="Pizza Place" />
    <pref keyword="mediterranean" type="Mediterranean Restaurant" />
    <pref keyword="burger" type="Burger Joint" />
    <pref keyword="hamburger" type="Burger Joint" />
    <pref keyword="bagel" type="Bagel Shop" />
    <pref keyword="bagels" type="Bagel Shop" />
    <pref keyword="dessert" type="Dessert Shop" />
    <pref keyword="bbq" type="BBQ Joint" />
    <pref keyword="barbecue" type="BBQ Joint" />
    <pref keyword="bar" type="Bar" />
    <pref keyword="bars" type="Bar" />
    <pref keyword="pub" type="Bar" />
    <pref keyword="pubs" type="Bar" />
    <pref keyword="alcohol" type="Bar" />
    <pref keyword="beer" type="Bar" />
    <pref keyword="wine" type="Winery" />
    <pref keyword="Tea" type="Tea Room" />
    <pref keyword="Trains" type="Train Station" />
    <pref keyword="books" type="Bookstore" />
    <pref keyword="reading" type="Bookstore" />
    <pref keyword="book store" type="Bookstore" />
    <pref keyword="book shop" type="Bookstore" />
    <pref keyword="vegetarian" type="Vegetarian %2F Vegan Restaurant" />
    <pref keyword="vegan" type="Vegetarian %2F Vegan Restaurant" />
    <pref keyword="donut" type="Donut Shop" />
    <pref keyword="donuts" type="Donut Shop" />
    <pref keyword="doughnut" type="Donut Shop" />
    <pref keyword="doughnuts" type="Donut Shop" />
  </types>
</parserData>
```
<pref keyword="asian" type="Asian Restaurant" />
<pref keyword="spanish" type="Spanish Restaurant" />
<pref keyword="middle east" type="Middle Eastern Restaurant" />
<pref keyword="middle eastern" type="Middle Eastern Restaurant" />
<pref keyword="ice cream" type="Ice Cream Shop" />
<pref keyword="eastern europe" type="Eastern European Restaurant" />
<pref keyword="eastern european" type="Eastern European Restaurant" />
<pref keyword="steak" type="Steakhouse" />
<pref keyword="portuguese" type="Portuguese Restaurant" />
<pref keyword="french" type="French Restaurant" />
<pref keyword="nightclub" type="Nightclub" />
<pref keyword="nightclubs" type="Nightclub" />
<pref keyword="art" type="Art Gallery" />
<pref keyword="art gallery" type="Art Gallery" />
<pref keyword="art galleries" type="Art Gallery" />
<pref keyword="museum" type="Museum" />
<pref keyword="museums" type="Museum" />
<pref keyword="jewellery" type="Jewelry Store" />
<pref keyword="sporting goods" type="Sporting Goods Shop" />
<pref keyword="spa" type="Spa" />
<pref keyword="cosmetics" type="Cosmetics Shop" />
<pref keyword="make-up" type="Cosmetics Shop" />
<pref keyword="makeup" type="Cosmetics Shop" />
<pref keyword="bike" type="Bike Shop" />
<pref keyword="bikes" type="Bike Shop" />
<pref keyword="cycling" type="Bike Shop" />
<pref keyword="music shop" type="Music Store" />
<pref keyword="music store" type="Music Store" />
<pref keyword="sweets" type="Candy Store" />
<pref keyword="sweetshop" type="Candy Store" />
<pref keyword="electronics" type="Electronics Store" />
<pref keyword="technology" type="Electronics Store" />
<pref keyword="yoga" type="Yoga Studio" />
<pref keyword="shoes" type="Shoe Store" />
<pref keyword="shoe shop" type="Shoe Store" />
<pref keyword="shoe store" type="Shoe Store" />
<pref keyword="arts and crafts" type="Arts & Crafts Store" />
<pref keyword="video store" type="Video Store" />
<pref keyword="video shop" type="Video Store" />
<pref keyword="videos" type="Video Store" />
<pref keyword="films" type="Video Store" />
<pref keyword="antique" type="Antique Shop" />
<pref keyword="game" type="Video Game Store" />
<pref keyword="games" type="Video Game Store" />
<pref keyword="video games" type="Video Game Store" />
<pref keyword="video game" type="Video Game Store" />
<pref keyword="computer games" type="Video Game Store" />
<pref keyword="gym" type="Gym" />
<pref keyword="gyms" type="Gym" />
<pref keyword="hardware" type="Hardware Store" />
<pref keyword="gardening" type="Garden Center" />
<pref keyword="cocktail" type="Cocktail Bar" />
<pref keyword="cocktails" type="Cocktail Bar" />
<pref keyword="hotels" type="Hotel" />
<pref keyword="hotel" type="Hotel" />
<pref keyword="scenic" type="Scenic Lookout" />
<pref keyword="nice views" type="Scenic Lookout" />
<pref keyword="good views" type="Scenic Lookout" />
<pref keyword="swimming pool" type="Pool" />
<pref keyword="swimming" type="Pool" />
<pref keyword="swimming pools" type="Pool" />
<pref keyword="golf" type="Golf Course" />  
<pref keyword="castle" type="Castle" />  
<pref keyword="castles" type="Castle" />  
<pref keyword="trails" type="Trail" />  
<pref keyword="walking" type="Trail" />  
<pref keyword="walks" type="Trail" />  
<pref keyword="farms" type="Farm" />  
<pref keyword="library" type="Library" />  
<pref keyword="libraries" type="Library" />  
<pref keyword="church" type="Church" />  
<pref keyword="churches" type="Church" />  
<pref keyword="religion" type="Church" />  
<pref keyword="history" type="History Site" />  
<pref keyword="historic sites" type="Historic Site" />  
<pref keyword="theater" type="Theater" />  
<pref keyword="theaters" type="Theater" />  
<pref keyword="american" type="American Restaurant" />  
<pref keyword="music school" type="Music School" />  
<pref keyword="pets" type="Pet Store" />  
<pref keyword="pet store" type="Pet Store" />  
<pref keyword="pet stores" type="Pet Store" />  
<pref keyword="pet shop" type="Pet Store" />  
<pref keyword="pet shops" type="Pet Store" />  
<pref keyword="sports" type="Sports Club" />  
<pref keyword="sports club" type="Sports Club" />  
<pref keyword="tanning salon" type="Tanning Salon" />  
<pref keyword="tanning salons" type="Tanning Salon" />  
<pref keyword="bowling" type="Bowling Alley" />  
<pref keyword="bowling alley" type="Bowling Alley" />  
<pref keyword="bowling alleys" type="Bowling Alley" />  
<pref keyword="mall" type="Mall" />  
<pref keyword="malls" type="Mall" />  
<pref keyword="shopping center" type="Mall" />  
<pref keyword="shopping centre" type="Mall" />  
<pref keyword="outdoors" type="Other Great Outdoors" />  
<pref keyword="park" type="Park" />  
<pref keyword="parks" type="Park" />  
<pref keyword="department store" type="Department Store" />  
<pref keyword="bed and breakfast" type="Bed &amp; Breakfast" />  
<pref keyword="camping" type="Campground" />  
<pref keyword="camping ground" type="Campground" />  
<pref keyword="athletics" type="Athletics &amp; Sports" />  
<pref keyword="football" type="Football Stadium" />  
<pref keyword="jazz" type="Jazz Club" />  
<pref keyword="concert" type="Concert Hall" />  
<pref keyword="concerts" type="Concert Hall" />  
<pref keyword="concert hall" type="Concert Hall" />  
<pref keyword="theater" type="Bank" />  
<pref keyword="bank" type="Bank" />  
<pref keyword="banking" type="Bank" />  
<pref keyword="casino" type="Casino" />  
<pref keyword="casinos" type="Casino" />  
<pref keyword="gambling" type="Casino" />  
<pref keyword="mosque" type="Mosque" />  
<pref keyword="mosques" type="Mosque" />  
<pref keyword="islam" type="Mosque" />  
<pref keyword="english" type="English Restaurant" />  
<pref keyword="science" type="Science Museum" />  
<pref keyword="movie theater" type="Movie Theater" />  
<pref keyword="movies" type="Movie Theater" />  
<pref keyword="cinema" type="Movie Theater" />
<pref keyword="zoo" type="Zoo" />
<pref keyword="zoos" type="Zoo" />
<pref keyword="animals" type="Zoo" />
<pref keyword="dancing" type="Dance Studio" />
<pref keyword="dance studio" type="Dance Studio" />
<pref keyword="entertainment" type="General Entertainment" />
<pref keyword="events" type="Event Space" />
<pref keyword="paintball" type="Paintball Field" />
<pref keyword="paint ball" type="Paintball Field" />
<pref keyword="gaming cafe" type="Gaming Cafe" />
<pref keyword="games cafe" type="Gaming Cafe" />
<pref keyword="sports bar" type="Sports Bar" />
<pref keyword="spiritual" type="Spiritual Center" />
<pref keyword="spiritual centre" type="Spiritual Center" />
<pref keyword="tattoos" type="Tattoo Parlor" />
<pref keyword="tattoo parlours" type="Tattoo Parlor" />
<pref keyword="tattoo parlour" type="Tattoo Parlor" />
<pref keyword="korean" type="Korean Restaurant" />
<pref keyword="breakfast spot" type="Breakfast Spot" />
<pref keyword="mexican" type="Mexican Restaurant" />
<pref keyword="sushi" type="Sushi Restaurant" />
<pref keyword="vietnamese" type="Vietnamese Restaurant" />
<pref keyword="whisky" type="Whisky Bar" />
<pref keyword="malaysian" type="Malaysian Restaurant" />
<pref keyword="tapas" type="Tapas Restaurant" />
<pref keyword="greek" type="Greek Restaurant" />
<pref keyword="portuguese" type="Portuguese Restaurant" />
<pref keyword="diner" type="Diner" />
<pref keyword="diners" type="Diner" />
<pref keyword="beer garden" type="Beer Garden" />
<pref keyword="beer gardens" type="Beer Garden" />
<pref keyword="brewery" type="Brewery" />
<pref keyword="breweries" type="Brewery" />
<pref keyword="lounge" type="Lounge" />
<pref keyword="gastropub" type="Gastropub" />
</types>
<states>
  <state keyword="favourite food" weight="likeHigh"/>
  <state keyword="favourite place" weight="likeHigh"/>
  <state keyword="favourite shop" weight="likeHigh"/>
  <state keyword="favourite" weight="likeHigh"/>
  <state keyword="love" weight="likeMedium"/>
  <state keyword="like" weight="likeLow"/>
  <state keyword="adore" weight="likeHigh"/>
  <state keyword="partial to" weight="likeMedium"/>
  <state keyword="fancy" weight="likeMedium"/>
  <state keyword="appealing" weight="likeMedium"/>
  <state keyword="can't get enough" weight="likeHigh"/>
  <state keyword="cannot get enough" weight="likeHigh"/>
  <state keyword="enjoy" weight="likeMedium"/>
  <state keyword="get a kick out of" weight="likeMedium"/>
  <state keyword="hanker for" weight="likeMedium"/>
  <state keyword="hankering" weight="likeMedium"/>
  <state keyword="desire" weight="likeHigh"/>
  <state keyword="indulge" weight="likeLow"/>
  <state keyword="relish" weight="likeHigh"/>
  <state keyword="revel" weight="likeHigh"/>
  <state keyword="savour" weight="likeHigh"/>
  <state keyword="interested" weight="likeMedium"/>
  <state keyword="interest in" weight="likeMedium"/>
  <state keyword="do not like" weight="dislikeLow"/>
  <state keyword="don't like" weight="dislikeLow"/>
  <state keyword="Hate" weight="dislikeHigh"/>
  <state keyword="detest" weight="dislikeHigh"/>
  <state keyword="do not love" weight="dislikeMedium"/>
  <state keyword="don't love" weight="dislikeMedium"/>
  <state keyword="do not fancy" weight="dislikeLow"/>
  <state keyword="don't fancy" weight="dislikeLow"/>
  <state keyword="not interested" weight="dislikeMedium"/>
  <state keyword="don't have an interest in" weight="dislikeMedium"/>
  <state keyword="do not have an interest in" weight="dislikeMedium"/>
  <state keyword="do not have any interest in" weight="dislikeMedium"/>
  <state keyword="do not have any interest in" weight="dislikeMedium"/>
  <state keyword="do not enjoy" weight="dislikeMedium"/>
  <state keyword="don't enjoy" weight="dislikeMedium"/>
  <state keyword="gone off" weight="dislikeMedium"/>
  <state keyword="uninterested" weight="dislikeMedium"/>
  <state keyword="can't stand" weight="dislikeHigh"/>
  <state keyword="cannot stand" weight="dislikeHigh"/>
</states>
Appendix E: Default User Model XML File

```xml
<xml version="1.0" encoding="UTF-8" standalone="no">
<UserModel>
  <Preferences>
    <Preference count="0" state="like" term="short term" type="cafe"/>
    <Preference count="0" state="like" term="short term" type="shop"/>
    <Preference count="0" state="like" term="short term" type="named building"/>
    <Preference count="0" state="like" term="short term" type="restaurant"/>
  </Preferences>
</UserModel>
```
Appendix F: Glossary

**Requirement:** A function or quality of the proposed system.

**Functional Requirement:** Functional requirements are the fundamental or essential subject matter of the product. They describe what the product has to do.

**Non-functional Requirement:** Non-functional requirements are the properties that the functions must have, such as performance and usability. These requirements are as important as the functional requirements for the product’s success.

**SpeechCity:** An Android application which allows hands and eyes free exploration of a city.

**Interaction Manager (IM):** SpeechCity’s Interaction Manager. Deals with requests and presents information to a user.

**City Model (CM):** A model of the city the user is in. This is Edinburgh throughout this project. Contains information on each location in Edinburgh such as; name, type, address, locations, etc.

**Pedestrian Tracker:** SpeechCity’s location tracker.

**User Model:** A model of a user based on the information which it contains. Used to personalise a system.

**User Model component:** The software to be produced for this project. The component of SpeechCity which will build and manage a User Model.

**User Model design pattern:** The method which is employed to create and populate the User Model.