The University of York Department of Computer Science

Submitted in part fulfilment for the degree of BEng.

Bringing Knowledge Through Machine Learning and SMS

Sam Heather
sam@heather.sh

28th April 2015

Supervisor: Dr. Lilian-JL Blot

Number of words = 23835, as counted by detex | wc -w.
This includes the body of the report, Bibliography and Appendices.

Abstract

Accessing information whilst without access to books or the Internet is near impossible, yet whether the situation is short term or long term it's common for an individual to be carrying a mobile phone capable of SMS. This dissertation aims to use SMS to create an application capable of automatically answering questions about geographical places. The project looks specifically at Natural Language Processing and Machine Learning to improve the quality of service provided over time.

A server application has been created which does this using various online data sources to locate answers to questions. The resulting application is tested against defined requirements and is used in an experimental evaluation. The experimental evaluation shows that the application is both useful and usable, and that the machine learning is effective at improving the quality of answers over time. Overall this dissertation looks at the process followed and the issues faced with developing this project, which resulted in an application that provides a useful access point to information.

Contents

List of Figures

List of Tables

1 Statement of Ethics

At the end of this project, an experimental evaluation was carried out which involved participants using the system developed and completing a questionnaire about their experience. Informed consent was given by each participant before they decided to partake in the experiment using a consent form. Participants were able to withdraw at any time without providing a reason and were reminded of this on the consent form. The consent form explained that the data would be collected from the participants and stated that they were able to withdraw their data after the experiment. It also informed participants exactly who would have access to the data collected and the format that they would have access to it in, specifically that it would be anonymised and kept confidential with the single exception of the Supervisor. In this case, participants' individual responses were kept confidential and their name was only collected on the consent form, after which a new identifying number was assigned to each participant which linked the participants' name on the consent form to the data collected. The consent forms were kept locked in a cupboard during the 4 days that the experiment was run, and upon completion they were transferred to the Supervisor for safe keeping in University facilities. The consent forms are the only method of identifying a participant. No participants were harmed by the experiment, and no participants under the age of 18 were used. A copy of the consent form, questionnaire and instructions for the experiment can be seen in Appendices B, C and D.

The root credentials for the server used to host the application were provided to the Supervisor in advance of any data collection. This allows the University access to all information in the system for data protection purposes. After the completion of the experiment a dump of the data in the database was also provided to the Supervisor for safe keeping.

Close attention to ethical issues, in particular privacy, was also paid during the design and development of the software. A key example of this is the use of phone numbers to uniquely identify users of the system. These are never directly stored in the database: instead only an obscured representation of the phone number is stored, which is generated using an industry standard one-way hashing function. This means that it is impossible to gain the identity of a user from the database, protecting their privacy in the event that the database is hacked. Additionally,

service providers who would carry information over SMS were researched to assess their reputability, as they would be transmitting information to private individuals.

Another ethical issue that attention was paid to was the potential for the application to cause harm. The choice of Geography as the subject for the software to answer questions on was made to remove the risk of the software working in a domain where it could cause harm. Answers for questions relating to Geographical entities are extremely unlikely to result in someone making a decision that could lead them harming themselves.

This system was only developed for research purposes, but if it were to be extended to be used outside of this context a feature should be built in to allow a user to delete all data corresponding to them via SMS to comply with data protection regulations. This was not done in this project as participants specifically gave consent for information to be collected and they will not have access to the system after the experiment as the server will be shutdown. Participants were provided with the Supervisor's contact details which they could use to request the deletion of their data in the future. During the experiment the phone numbers of participants were not collected or used.

Finally, it was noted in early research that there is an ethical responsibility to only provide correct information to a user in response to a question. An attempt is made to significantly reduce the chance of incorrect information being sent to a user by assessing the quality of a potential answer and checking that it is above a quality threshold. This check compares the information that has been returned to the information requested by the user to make sure that they are related, and only if it passes this test is the answer sent to the user.

Acknowledgement

The author would like to thank various individuals for their contributions to this project. First, the author would like to thank Lilian Blot for his time and effort spent supervising and providing guidance for this project. He would also like to thank Julie Markham and Nicholas Hopper, with whom the author collaborated with on Shy, the mobile application that inspired this project. Finally, the author would like to thank Emma Rizkallah and Ashley Clayton for providing guidance on writing style and reviewing this report.

The author would also like to acknowledge the organisations who supported this project. These were SpazioDati (http://spaziodati.eu/), who providing extended access to the Dandelion Text Semantics API at no extra charge, and Rackspace who provided free server infrastructure.

2 Introduction

2.1 Background of this project

The inspiration for this project originates from a project the author undertook in September 2014 whilst attending Yacht Hack, a week-long hackathon on a Yacht in Croatia [**?**]. The author co-created a project called Shy to prototype a mobile application that would facilitate the immediate answering of questions that fall within certain categories using a knowledge base, simple machine learning and profiling of users based on their usage history.

One of the initial goals of Shy was to bring knowledge via m-learning[1] as opposed to e-learning[2]. This is because m-learning systems are available to a much larger demographic of people as a result of the hardware (mobile devices, commonly phones) becoming cheaper and more accessible.

A semi-functional prototype was completed for iOS, as seen in Figure 2.1. The front-end was complete; however in the backend question-answer matching was evaluated without knowledge of previous material the user had viewed. This meant that explicitly targeted answers for a question and suggestions of questions that a given user profile might be interested in could not be made. Up until this point, the service was restricted to working on iOS smart devices only, with poor quality question-answer matching. In the initial project, the knowledge base was built for questions on personal health, sex, relationships and family.

2.2 Motivation for this project

Access to knowledge is a critical part of modern life. It's also a Human Right, under Article 27 of the Universal Declaration of Human Rights [**?**]. As members of a developed country, we have the facility to retrieve information and instantly communicate with each other using our smart phones, and commonly do it up to

[1] M-Learning: learning using tools available on a mobile device

[2] E-Learning: learning conducted online, usually with the help of a computer

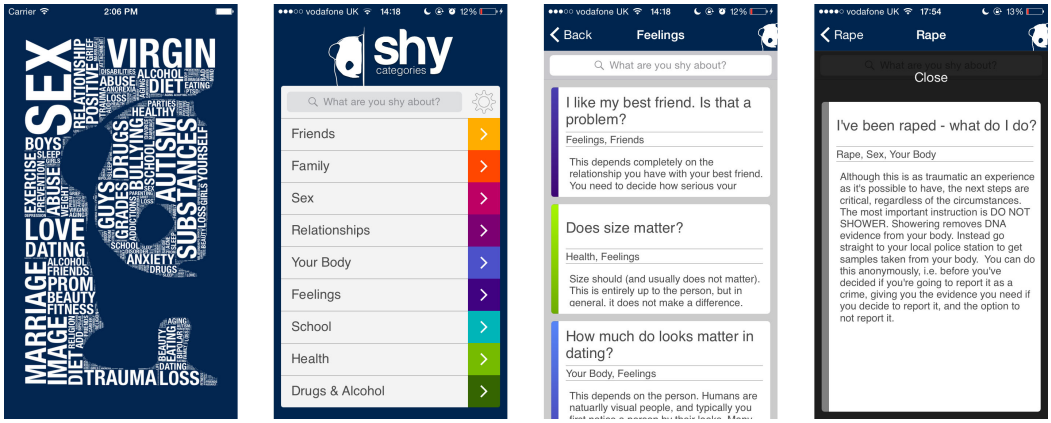

Figure 2.1: Screenshots of Shy iOS App

200 times a day [**?**], which is a facility that hundreds of millions of individuals in developing countries live without. This project aims to investigate the technical and ethical challenges behind this and to create a technology to give more people access to knowledge, and thus their human rights.

The project involves the use of a number of systems, including machine translation, an SMS input/output system and a system to build a profile on a user in order to facilitate high-quality targeted question-answer matching. The system will also feature machine learning so that as time progresses, the quality of the answers it returns increases. The use of these systems in one application along with complex ethics and privacy issues create an interesting project that draws together many technologies and discussions in a way that can be used as a basis for further research in the future.

2.3 Aims of this project

The first goal of this project is to assess and address the ethical issues that arise from the software that will be created. These include the responsibility that the software and its developer has because of its position of trust to provide accurate information and to protect the privacy of its users by storing only necessary information, among others. This project will do this by researching ethical issues relating to the technologies that the project will use, for example machine translation and user information storage. This will then be used to feed the design process of the software and to specify the expected use-case of it. Finally, the resulting software will be evaluated through experimentation, with volunteers asking the software a

set of questions within a topic area, in a non-English language, and evaluate the relevance of answers returned.

Initially, the goal for this dissertation was to focus on questions relating to personal health, sex, relationships and family, which was as a direct result of the focus of 'Shy'. Although a health and diseases database was located in the early stages of the project (Diseases Database, www.diseasesdatabase.com), it was targeted at professional uses for the data and came with a significant licence fee, as well as complex documentation that the provider customised specifically for each use-case. An attempt was made to acquire a free academic research licence via email but this was unsuccessful. Additionally, as the project progressed in the early stages it became clear that the author's original two-page document discussing ethical considerations with the project did not cover all possible situations. Indeed, further significant ethical issues were still been discovered as far as eight weeks into the project, which would have required a complex ethics panel review, critically delaying this project. One major issue was that any application that dispenses health related information that a user might act upon in a life changing manner has to always provide correct information in response to a query, since an incorrect response could lead to a user taking an action which leads to them harming themselves. No information source containing knowledge of a high enough quality with a licence that allowed it to be used in this project could be found. Another issue was that privacy, which is critical to a project in the area of personal health, couldn't be guaranteed since information would be transmitted in an unencrypted form via SMS.

As a result of the above issues with locating a reputable knowledge base and the complex ethical issues, a decision was made in early December 2014 to switch the target topic for questions to Geography and to use Wikipedia as a datasource, since it is openly available, well populated and multiple methods exist for extracting information from it. In future work, it should be possible to extend this project to work with medical questions, if a high quality data-source is located and a solution to the ethical issues is found. A full discussion of possible future work is included in section 9.

2.4 Structure of this report

This report starts with a literature review in chapter 3 where existing software, tools and services of a similar type to those that will be used in this project are presented, along with research into the ethical issues of the creation and use of the software under development. Comparisons between different software development life-cycles are also described and discussed.

Chapter 4 describes the method that will be taken to develop the software and service. This covers the software development life-cycle that will be followed, and sets out a plan for when development will take place. Requirements will also be identified, described and categorised as either functional or non-functional. A method of evaluating the success of the software will also be discussed and chosen.

The design of the software, driven from information collected from the requirements, will be set out in Chapter 5. The individual components that make up the technology will be described and discussed, followed by a set of annotated diagrams showing how these modules will interface with each other to build a complete system. Finally, this section will include a description of the platform, language and other tools that will be used.

Chapter 6 contains details about the implementation of the software. This includes information on the tools that will be used to create components such as the SMS interface and question-answer matching system.

Results from the evaluation of the system will be presented in Chapter 7, where they will also be discussed and their implication on this project given.

Chapter 8 contains an evaluation of the software produced and the process taken to complete it. Comparisons will be made to the success criteria, set out for the project in section 4.5.

Thoughts on future research that builds on the work produced in this project are then displayed in chapter 9.

Finally, the main achievements of this project are concluded in chapter 10 with closing thoughts.

2.5 Assumptions Within This Project

This project will make the assumption that users of this service have basic literacy skills in a language supported by the project. Although this is not a correct assumption world-wide, expanding the remit of this project to include a 'graphical' user interface is not possible as SMS only supports the transmission of short pieces of text.

The main use-case for this project is going to be for people in parts of the world with limited internet access, whether this is a short term situation or a long term situation. Although these places exist all over the world, the African continent is typical of this problem so I will focus my research in demonstrating a need for this tool there.

Another assumption made in this project is that the users who may use this service have access to SMS facilities. Although SMS coverage does cover a majority of the world there are remote areas where it does not, and in these areas the application would be inaccessible. Another consequence of this is that the poorest of users, those without a non-smart phone, will not be able to use the application developed.

Finally, an assumption is made about the types of question that a user may wish to ask of the application. It is assumed that users will only want to ask questions in two forms: a question about a specific property of a geographical entity; and a request for general information about a geographical entity. This assumption is made to simplify the implementation to allow it to be created by a single developer in the course of this project. The consequence of this is that the application will not be able to handle more complex questions.

3 Literature Review

A significant amount of work has previously been undertaken in the area of making knowledge quickly accessible to us in the form of questions and answers. To take one example, imagine that an individual needs to find the population of London as quickly as possible. The initial step would be to search for information. In the connected world, this is easy: a simple Google search returns the answers without even having to click through to any resources, as shown in Figure 3.1. It should be noted that Google do not provide any information on how they do this or the data sources that they use.

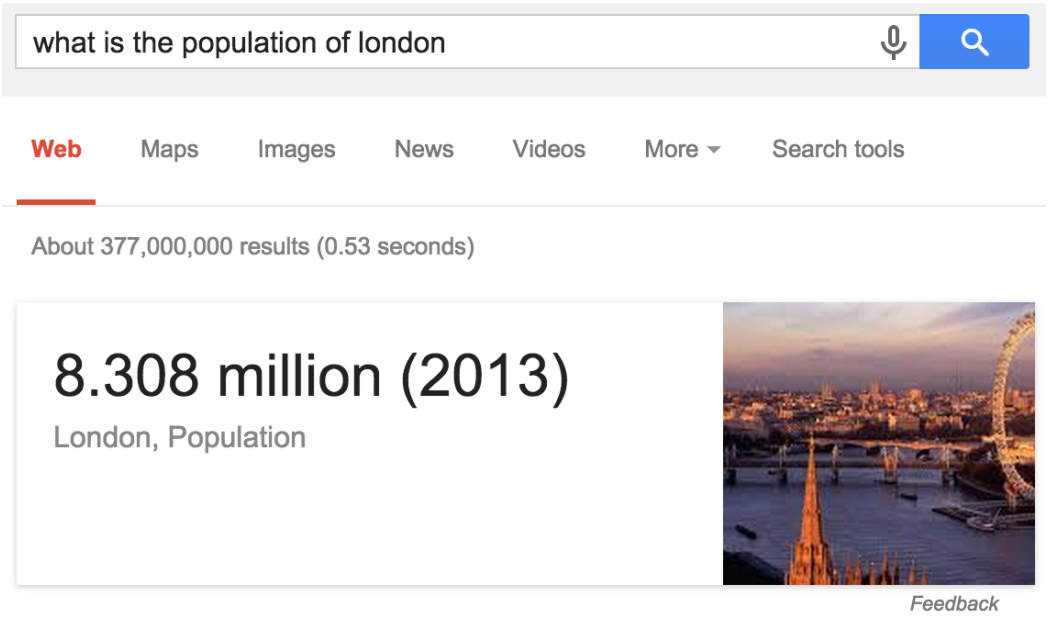

Figure 3.1: Google Now information box. Screenshot taken on 3rd February 2015.

This is easy to do in the developed world where up to 75% of the population are connected to the internet [**?**], with the population of elderly people being largely responsible for the remaining 25% [**?**]. This contrasts strongly with the situation in the African continent, where in 2013, internet usage had only reached 16%, a figure largely inflated by South Africa where 5% of the African population generate

two-thirds of internet traffic from the African Continent [**?**].

3.1 Previous work using SMS

Although the above suggests that the African continent is majoritively disconnected, this is not the case. Because of restrictions in the electricity available, the ways in which a non-smart phone can be used in Africa have far surpassed those in the developed world [**?**], in part due to the longer battery life of a non-smart mobile phone compared to a smart mobile phone. Interesting examples of SMS use in Africa include an automated service which sends an SMS to HIV/AIDS sufferers to remind them to take their medication, and a system for farmers that gives them the current market prices for goods, saving them from making regular long-distance trips to the market or relying on out-of-date information from a weekly radio broadcast [**?**].

Interactive systems have also been developed to operate over SMS. One successful example is mobile money platforms. These allow for users from any background to pay and be paid for goods and to transfer money across long distances at negligible cost [**?**]. One of the most highly adopted services is M-Pesa, which in Kenya alone was responsible for £5.7 Billion in transfers in 2012 [1]. M-Pesa gives users a balance linked to a national ID number from which they can pay for goods by sending an SMS with a cashier (recipient) number or pay outstanding bills in a similar way. Non-subscribers can also use the system by depositing money with a M-Pesa cashier in exchange for an access code, which can be sent via SMS to a contact who can subsequently redeem it with their local cashier [**?**].

This difference in standard use of mobile phones is demonstrated in the International Telecommunications Union's 2013 report [**?**], which shows that in Europe, for 790 million mobile subscriptions, 53% of subscribers have mobile internet access (422 million), compared to 17% in Africa (93 million have mobile broadband, out of 545 million mobile subscribers). This is due to the prohibitively high cost of accessing data services, regardless of the hardware that the user has. In 2012 in Europe, 500 MB of data per month for 12 months cost 1.2% of the average Gross National Income Per Capita (GNI pc). In Africa, the average price was 30 times this, at 36.2% of an individuals GNI pc [**?**].

[1] Data from Safaricom, M-Pesa operator in Kenya - actual value 817,085,000,000 Kenyan Shillings, converted to GBP on 1st November 2014 at rate of approximately 0.007.

3.2 Machine Learning Techniques

There are two common machine learning paradigms which will be described in this section. These are Supervised Leaning and "Learning without a teacher" [**?**]. Supervised learning is the paradigm of machine learning techniques that involve learning from a 'teacher', where the teacher gives the exact expected output for a possible input. The application is then able to learn from this output, usually by comparing to the output that it would have produced previously, thus improving the output in future use. [**?**].

The paradigm of Learning Without a Teacher is split into two subdivisions. The first of these is Reinforcement Learning [**?**], which shares some characteristics with supervised learning. Instead of using a teacher that provides the correct output for a given input into the system, the feedback comes as a yes or no signal intended to reinforce whether the output the system provided was correct or not. This system is binary and so doesn't provide information on what the correct output should be and how much the system needs to change by to produce the optimal output [**?**].

The second subdivision of the "Learning Without a Teacher" paradigm is that of Unsupervised Learning [**?**]. Unsupervised learning is where there is no teacher or other source of information about the ideal results of the system. The system typically has an internal method for establishing the quality of an output, which it then uses to adjust and improve [**?**].

The goal of this application is to use a Machine Learning technique to improve the quality of answers, based only on the feedback received from the user. Supervised learning techniques cannot be used as it is not feasible for a user to act as a 'teacher' and train the system by providing a sufficient number of training examples to identify a pattern that will be useful for all possible questions. However, a user is however able to provide feedback on whether an answer is of high or low quality, thus fulfilling the requirement to use Reinforcement Learning. With regard to Unsupervised Learning, to use this approach a automated system would have to be developed to provide feedback to the Machine Learning. This system would have to be capable of identifying if an answer was correct for any question, which is out of the scope of this project.

One common technique for building a Reinforcement Machine Learning system for learning if an output was of high or low quality is to use "Learning by Reward/Punishment" [**?**]. In this technique each concept or rule is assigned a weighting initially. Then, if the system receives positive feedback for an outcome it increases the weighting for that concept or rule. If the system receives negative feedback for an outcome it decreases the weighting for that rule or concept [**?**].

3.3 Semantically Comparing Text

This project will use semantic comparison of text to locate information in remote data sources, so it is important to have an understanding of the different methods and measures for doing this.

Semantic similarity is a measure of how semantically alike entities are in their meaning, for example, `bank` and `trust company` [**?**]. A more general measure exists called Semantic Relatedness. Whereas semantic similarity compares entities by their meaning, entities may be semantically related by additional lexical relationships such as antonymy (the opposite of an entity, for example, `tall` and `short`) and meronymy (entities that are a part of another entity, for example, `car` and `wheel`) [**? ?**].

3.4 Ethical Issues

This project raises a number of ethical issues related to translation accuracy, providing information to people in an ethically acceptable way (with a focus on information accuracy) and maintaining user privacy.

3.4.1 Ethics of Providing Information

One issue for this project comes from providing information that may affect an individual or lead to them to take a harmful action. In a similar way to that which a teacher has a responsibility to teach accurate information to a pupil due to their position of trust, any service relied upon by a user must equally provide accurate information. An analogy can be drawn between the application been developed and a teacher in modern life with regard to ethical issues about providing information. In modern society a teacher has a responsibility for directing the learning of students and ensuring that the material that students learn is accurate information [**?**]. The American Academy of Pediatrics describes a similar view; that Schools exist to educate people, not to misinform them [**?**]. In an example, they describe the situation of the Department of Health in Rhode Island changing the curriculum on sex education to teach abstinence instead of methods to have safe sex. The effect of providing this information to students in the past has been that the rate of sexually transmitted diseases and teenage pregnancies has increased as teenagers then do not have the knowledge that will prevent the unintended consequences of sex [**?**].

This clearly shows that as the application been developed in this project will be providing information from a position of trust (users will ask questions and then use the output), the information it provides must be correct if that information could result in a user taking a potentially harmful action.

3.4.2 Ethics of Translation

A significant part of this project is represented by the support of multiple languages. It is clear that translation on demand, at scale, needs to be automated by some kind of machine or algorithmic translation.

This project involves two blocks of translation. These are:

- The translation of the user's input into the language of the system (English)
- The translation of the answer to a user's question from the system language to their local language.

The latter of these raises some issues that do not apply to the former. To understand these issues it is first necessary to understand the two categories of machine translation.

Rule-based machine translation effectively treats human language in a similar style to programming languages. Formal grammars and lexicons are used to represent words that exist in either the source or target language, structures representing the translation of individual words or groups of words. Map structures contain mappings between individual or groups of words to their translated counterparts, sometimes with multiple results (a one-to-many map), from which rules decide which is selected. Maps and rule sets are created by trained computational linguists [**?**].

More commonly, Statistical Machine Translation (SMT) is used, for example, by Google and Microsoft in their translation products [**? ?**]. SMT learns mappings between strings of words of potentially unequal lengths from pre-existing original texts and their trusted human translations. The accuracy and breadth of language support for translation increases as more source material is analysed by the system, as potentially erroneous or low quality translations can be identified and flagged. SMT is also dependent on the quality of the human translation on the input material [**?**].

A significant issue that is present in statistical machine learning systems comes from the knowledge we assume an individual has and derives from a word [**?**]; in this project, this affects the translation of the answers returned to a user. In human

translation this is solved by the translator's knowledge of the difference in material culture, allowing them to append necessary information to the resulting translation that the recipient might find useful [**?**]. In statistical machine learning, cultural awareness of material knowledge is a separate problem on its own [**?**]. To take one example from Melby (2006):

> "when translating a French menu, a human translator might stop to think that an English speaker in France would appreciate being told that a steak tartare is served entirely raw, even if this information is not contained in the original text (because French people might be assumed to know this already). Such a translator would be aware of differences in material culture, and would be able to empathise with the English speaker who might choose to avoid the dish, given more information" [**?** **?**]

This issue is a result of the translation engine not being capable of taking and using a complete representation of the expected cultural differences between those who speak the input language and those who speak the output language [**?**]. This does not apply to the first block of translation within the application where a user's input is translated into the language of the system, since the application will only need to parse text to identify geographical entities and the property been queried, making context unnecessary.

With the current focus of the application being Geography it is unlikely that the information distributed will be used by individuals making important decisions so this issue is not relevant. However, in the future if the application is extended to work in more sensitive subject areas such as personal health, where users may make important personal decisions based on the answers returned, cultural confusion such as this could have potentially catastrophic effects.

3.4.3 User Privacy and Data Protection

This project stores a record of information that has already been returned to a user to allow it to handle questions based on previous questions and the software needs a method for identifying users to match them to their history. This raises the ethical issue of privacy. All information on a user has to be kept securely in a way that an individual user can not be identified.

One way of doing this is using a technology called hashing. Hashing is the technique of taking data as an input and generating an output value (called a 'hash') that is unique to an input [**?**]. Hash functions are ideally one way functions where given a piece of input data X, the output hash will always be a constant value Y,

userId	userData
8f64B2	{'previousQuestions':[13,301,170,577]}
9ac5e3	{'previousQuestions':[441,56]}
f9dd7e	{'previousQuestions':[301,623,89,280,364,621,209]}

Table 3.1: Example of software database: hashed phone numbers paired with a list of previous returned answer IDs.

whenever and however the hashing algorithm is executed [**?**]. This allows a user's identifying information such as their phone number to be obscured, in such a way that the mobile number can not be recovered from the database. When a question is received, the phone number (represented by variable X in the above example) can be hashed and the data for this number looked up from the database without the database application ever being aware of the phone number of the user. In the example in Figure 3.2, a phone number (X) is used as input to an imaginary hashing function and the output hash (Y) is shown on the right hand side. As just described, this number can then be used as the key for the user information database, shown in Table 3.1 to retrieve a users' information.

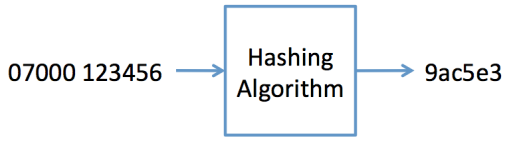

Figure 3.2: Hashing a phone number using an arbitrary hashing algorithm

In practise it is possible for the output of a hashing algorithm not to be unique to a particular input. This is known as the collision resistance property of a hashing algorithm and formally is the property of a hashing algorithm that determines the probability of the same output hash for two distinct random inputs [**?**]. It occurs because the range of possible input values is much larger than the range of output hashes, and because all valid inputs must map to an output there is repetition in outputs [**?**]. This property of hashing algorithms can be a security risk for some use-cases, where hashes are used to verify that data has not changed since been hashed. In this case, a third-party may be attempting to generate alternative, malicious data with the same hash as some pre-existing data, to swap them undetected [**?**]. In the use-case of this project there is no security risk since hashes are only used to anonymise phone numbers; however a high collision resistance is necessary to ensure that two users of the system with distinct phone numbers don't get mapped

to the same row in the database, resulting in poor quality question-answer matching or in the extreme worst case, privacy issues originating from returning to one user answers that are targeted to another.

Irreversibility is a necessary property of the hashing algorithm chosen for this project. This ensures that the key used in the database cannot be used to retrieve the phone number of the user, thus protecting their identity. Commonly used hashing algorithm families, such as the MD family (e.g. MD5) and SHA family are all designed to be one-way [**?**].

Although hashing algorithms are computationally one-way, they can suffer from another type of vulnerability affecting the security of the original input data. When hashes are used for short strings of information (passwords, for example), a simple way to try and retrieve the original data is to compute a dictionary of the hashes of lots of common passwords, from which matches can be found. This idea has been extended to produce Precomputed Hash Chains and subsequently Rainbow Tables. These are highly efficient data structures that consist of chains of processed input messages and their hashes with a reduce function to reach the next item in the chain, allowing original input data to be looked-up from its hash. Both Precomputed Hash Chains and Rainbow Tables tend to be Terabytes in size, and as such they only exist in a significant form for the most commonly used hash functions (including MD5 and SHA) [**?**]. Within this project, this raises privacy implications, as hashes within the database would be 'convertible' to phone numbers.

In password authentication systems, this can be solved by the use of a technique called Salting [**?**]. Salting is where a 'salt' - a piece of additional unique data - is added to each piece of input data before hashing, and then this salt is prepended to the output hash. This nullifies the use of Precomputed Hash Chains and subsequently Rainbow Tables [**?**]. When checking if a plaintext password is equal to the hashed and salted representation, the salt from the stored hash is added to the plaintext password, the hashing algorithm applied and then the output compared to the stored obscured password. However, in this application this would prevent efficient lookup of users from phone numbers in the application, as the application would have to compare an input phone number against every user object in the database, which is computationally expensive and slow since a salted hash has to be computed for potentially every user record [**?**].

As such, it has been decided that the best way of protecting a user's phone number in the event of a database hack without significantly degrading performance is to salt phone numbers with a constant, large and complex salt. This means that a rainbow table would have to be specifically computed specifically for this application and would be extremely large in size.

3.5 Software Design Life Cycles

There are many software development life cycles that a developer or company has available to choose from. Four common ones include the Waterfall Model [**?**], Iterative Model [**?**][2], Spiral Model [**?**] and the Agile Development Model [**?**][3]. In order to choose which life cycle suits this project best, research into all four has been carried out.

3.5.1 The Waterfall Model

The waterfall model consists of 5 phases of work, where each leads directly into the next. It is a plan driven development model, which means that all process activities such as the implementation of the project must be planned and scheduled before work on them begins [**?**]. The five layers are shown in Figure 3.3 and described below:

1. **Requirements**. In this stage, both user and system requirements for the project are determined after examining the business goals for the project. These form the system specification.
2. **Design**. Here, a plan for the software development is created. The overall architecture for the software is created, the external libraries to be used are chosen and the interaction between different modules is modelled.
3. **Implementation**. In this stage, the software is implemented to match the design and architecture set out in the Design phase. Unit tests are written to ensure that each module matches its specification.
4. **Testing**. In this section the system is tested as a whole to check for any bugs that might not be picked up when testing individual modules in isolation. The software is then delivered.
5. **Maintenance**. Finally, the software is deployed and maintained. This includes making changes to the requirements and subsequently the software implementation as the business goals change. [**?**]

[2] Iterative development came together from many concepts dating back to 1957. There was no single paper that initially presented the entire concept, but this paper gives a summary of how the various concepts came together into the current Iterative Development model.

[3] The Agile Manifesto brought together a group of Agile Methods which had been in use since the mid-1990's into a single concept, that of Agile Development.

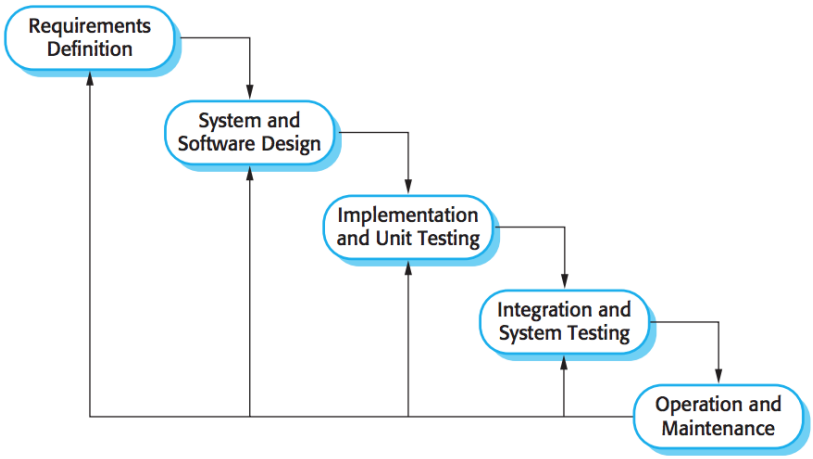

Figure 3.3: Waterfall Development Model [**?**]

A key advantage of the Waterfall Model is its simplicity: it is easy to understand and to use, saving time at the start of the project that would otherwise have been spent on learning and preparing to use a more complex model. The main disadvantage of the waterfall model is that once development has moved from one stage to another it cannot go back because the model is fixed and progresses in one direction only, until the maintenance step.

3.5.2 The Iterative Model

Iterative development is the practise of splitting the overall development of a project into multiple independent and distinct blocks. A block contains a sequence of tasks which are essentially a mini life cycle, such that each iteration can really be thought of as its own project. Each iteration tends to focus on a specific set of features such that when the iteration is complete the overall project is stable and the modules developed to date can be tested as a whole system [**?**].

There are a multiple variations of the iterative development method. Two of these are Timeboxed Iterative Development where all iterations have a fixed duration, and Client-driven Iterative Development where each iteration contains the features that the client considers have the highest business value [**?**].

A common variation is based on the waterfall model as shown in Figure 3.4. The first three tasks in the cycle represent the first four stages in the waterfall model (Requirements, Design, Implementation and Testing), at which point there is the option to either evaluate progress so far and start another iteration, or to deploy

the software and end development. One significant difference between this and the waterfall model is that maintenance is not represented.

A key advantage of the Iterative Model is that it allows for the user to revise their requirements as they progress, making it dynamic and flexible which is ideal for small development teams. This is also its disadvantage, in that the lack of fixed structure can be a weakness unless the project is planned well.

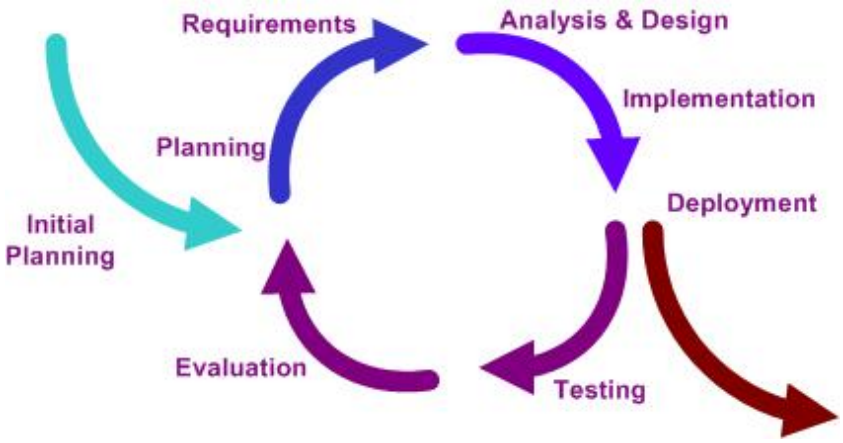

Figure 3.4: Iterative Development Method
source: https://commons.wikimedia.org/wiki/File:Iterative_development_model_V2.jpg

3.5.3 Boehm's Spiral Model

The Spiral Model [**?**] was originally developed as a result of the adjustments that were commonly made to the waterfall model in large government projects. The Spiral Model is a general model and can be used as a generator for other more specific models, given the parameters for a project such as its risks [**?**]. Models such as the waterfall model are specialisations of the spiral model [**?**].

In the Spiral Model each loop represents a stage of the software development, from planning on the inside loop to system testing and verification on the outside, as shown in Figure 3.5. The cyclic nature of the model reflects the evolution of development of a software engineering project: the displacement from the origin shows an increase in the definition of the software, progress with the implementation and a decrease in the overall risk [**?**]. With each spiral cycle in the model, stakeholder objectives are reviewed, risks identified and stakeholder approval and commitment sought before commencing the cycle [**?**].

A key advantage of the Spiral Model is the level of detail it models. The stage

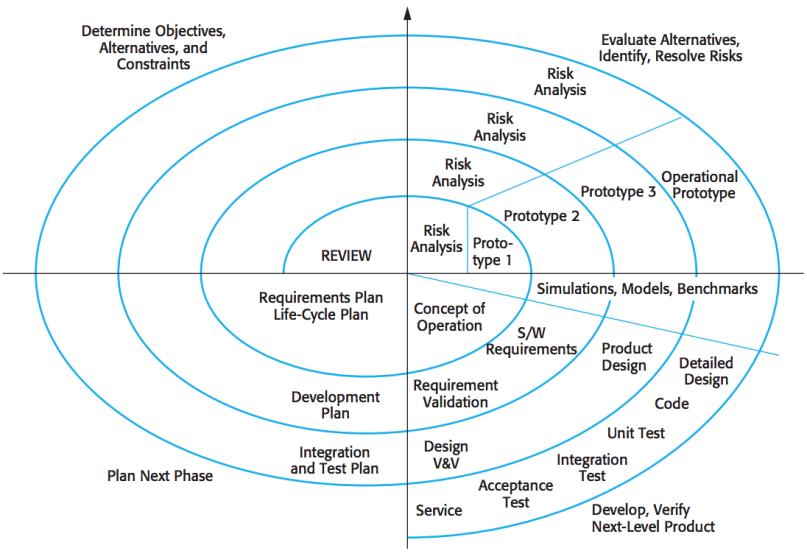

Figure 3.5: Boehm's Spiral Model [**?**]

of a sub-task within the entire development process is represented by the distance from the origin and each loop is split into small, well described sections. The main disadvantage is that the planning and use of the model is largely driven by the results of complex risk analysis on the sub-tasks [**?**]. Although this makes it favourable to large organisations, it has the opposite effect for smaller projects.

3.5.4 Agile Development

Agile Development essentially represents a group of development techniques targeted towards either the development of a small to medium sized product for sale or the development of a custom system within an organisation, where the customer has a high degree of involvement, for example 'Extreme Programming' [**?**].

Agile development is highly targeted towards teams with components such as Pair Programming, Collective Ownership, regular meetings and employee health (avoiding overtime) playing an integral part in the model [**?**].

3.5.5 Development Life Cycle Summary

Iterative development has been chosen as the model that this project will use, based on the following factors:

1. Boehm's Spiral Model and Agile Development are both targeted towards development teams. A significant part of these models relates to team interaction, so they are not appropriate to this project.
2. The Waterfall Model is rigid and inflexible. It does not allow for the changes to the requirements and implementation that might be proposed after evaluating the quality of the first implementation.
3. The Iterative development model can be implemented effectively by a single developer, and supports the adjustment of requirements and implementation in each iteration of the cycle.

4 Method and Requirements

In this chapter the method that will be followed for software development and an experimental evaluation of the software will be presented. The tools that will be used in the software will also be described.

4.1 Plan for Software Development

The Iterative Development Model was selected in Section 3.5.5. Therefore, the development of this service will follow the principles set forth in this process. Software development will take place in stages representing each iteration of the iterative process. After each iteration the software will be discussed in a meeting, with a focus on the developed capabilities, quality and any issues. Three iterations were scheduled that mapped to calendar months with the goal of having completed the software by the end of March.

Software development began in late December 2014 and the first iteration is expected to be complete by the end of January which will demonstrate matching a variable directly by exact name to the Wikipedia API. A testing application will also be developed to aid development. The following additional iterations are planned, although it is expected they will evolve as the software changes and more clarity is achieved with regard to the goals.

During the second iteration running through February semantic analysis and matching of requested properties / keys in the data sources will be added to the application, and additional data sources will be researched and included.

In the final iteration running through March a natural language processing module will be constructed to handle questions in natural form as opposed to using a set syntax as had been used during development. Any additional tools needed to run an experimental evaluation of the software will also be developed.

In the initial specification of the project one aspect was to make the software developed capable of handling queries in multiple languages. However it has been decided after researching the issues of machine translation in section 3.4.2 that

compatibility with multiple languages will be omitted from the software developed due to its complexity and the time constraints on this project. A discussion of how it can be added into the system in the future is provided in section 6.9.

4.2 Requirements

4.2.1 Use Cases / User Goals

Two personas were created to represent individuals who may have a need for the work resulting from this project, which can be seen in Appendix A. These are Nanjala, who is from Uganda, and Cedric, who is from Switzerland, and it is from these that user goals are extracted. These describe what a general user aims to achieve with this service.

1. Nanjala wants to be able to read general information on any given topic.
2. Cedric wants to be able to retrieve facts from direct questions.
3. Both Nanjala and Cedric want to be able to achieve their goals entirely via SMS.
4. Cedric needs a simple and efficient user interface that allows him to retrieve facts quickly.
5. Cedric wants to be able to feedback into the system when he finds an answer particularly useful or unhelpful.
6. Nanjala wants to be able to show the service to her friends without explaining how to use it every time.
7. Cedric wants the system to protect his privacy, obscuring his identity.
8. Nanjala wants to be able to ask questions as she would ask a friend without having to learn a syntax.

4.2.2 User Requirements

User goals are distilled into formal user requirements. Each user requirement corresponds directly to the correspondingly numbered user goal.

1. Users must be able to request general or descriptive information on a geographical entity.

2. Users must be able to ask questions requesting a fact or statistic about an entity and get the answer or number they are looking for.
3. Users must be able to interact with the system over SMS.
4. Users must be able to make their queries and get responses quickly.
5. Users must be able to feedback to the system when an answer is of noticeably low or high quality so the system can learn and improve.
6. Users must be able to learn how to use the system themselves and this learning process should be quick.
7. Users must be comfortable that their privacy is been protected whilst using the service.
8. Users must be able to interact with the system using natural language.

4.2.3 System Requirements - Functional

System requirements are derived from the capabilities that the software needs to have to be able to support the user requirements. To support the user requirements in section 4.2.2, the application system must fulfil the following system requirements.

1. The system will be capable of processing queries in natural language form.
2. The system will receive queries via SMS, process them and then respond via SMS.
3. The system shall use a datasource that contains both facts and descriptive text on geographical entities to answer queries.
4. The system must be able to parse queries to identify the entity that is being asked about and the specific parameter of that entity that should be returned as the answer to the user's query.
5. As well as giving direct facts and statistics, the system should be able to give general descriptive information on an entity.
6. The system must be able to match words of similar meaning to the parameters in the data source. For example, the word 'long' must be matched to the parameter 'length'.
7. The system must learn from user feedback when answers that are returned are of high or low quality and apply this knowledge to future queries.
8. The system must be self-explanatory, with the facility to offer instructions to

a user.

9. The system must use user state so that it can accept queries that are dependent on past questions, for example a query asking for more information or feedback on the quality of a past question.
10. The system must be easy to implement and flexible, to accommodate future projects that make use of adaptations of this project.
11. The system must obscure a user's phone number in the database.

4.2.4 System Requirements - Non-Functional

Non-functional requirements are listed in table 4.1 and are accompanied by a Fit Criterion, which will be used to identify whether a requirement has been met successfully.

4.3 Data Sources

A number of possible datasources were identified from which data could be drawn from for this project. These were all based on Wikipedia (a large, linked source of knowledge), and include the Wikipedia API and various representations of the DBPedia Ontology. In this section, both will be discussed and a decision will be made with regard to which ones will be used.

4.3.1 Wikipedia API

The Wikipedia API is a standard MediaWiki API and allows applications to query a page or resource for various parameters from a page, including the page contents, infobox parameters, citation list and links to other articles in various flat or multi-layered data structures [**?**]. Interactions are made via a HTTP GET request to a URL containing the parameters, and responses can be received in JSON. This makes the API easy to use and widely compatible, especially since JSON can be converted to a dictionary in most languages.

ID	Non-Functional Requirement	Fit Criterion
NF.1	The system behind the SMS gateway must respond in a timely manner.	From an SMS been received, a response must be sent within 20 seconds. SMS transmission times on the cellular network and gateway are out of the control of this project, so are not accounted for.
NF.2	The answers returned must be of high quality.	In testing, the system returns the correct information for at least 90% of queries initially (before learning starts) if an answer is returned at all.
NF.3	Answers that are consistently rated poorly are are less likely to be returned.	In testing, after receiving a small number of negative ratings for an answer, a different answer is returned.
NF.4	Instructions to use the service should be clear and concise, with no ambiguity.	In testing, having explained the purpose of the system, a user should be able to use the system with just the help information, without needing to ask for assistance.
NF.5	The system should accept and be able to process questions in natural human language as opposed to a delimited message.	In testing, a user should be able to use the system for general questions without having had a syntax explained to them.

Table 4.1: List of non-functional requirements and their Fit Criteria.

4.3.2 DBPedia Ontology

The DBPedia Ontology is the result of a project that aims to build an OWL ontology representing all of the knowledge within Wikipedia. The ontology has been manually created from the most commonly used infoboxes on Wikipedia and covers 685 classes with 2,795 properties, providing structured information on 4.2 million places, people, work, species and organisations [**?**]. This makes it especially good for use in applications since the data is both curated and well structured.

There are two ways of querying the DBPedia Ontology. One is using the SPARQL language to query the ontology via its SPARQL interface [**?**]; however this is complex as queries must be structured correctly and it is necessary to know the class structure of the entities being queried [**?**]. Matching a question asked by a user to entities within the Ontology is difficult and outside of the time scope of this project.

In the event that the class structure is not known, queries can be made by querying the top-level representation of an entity in the ontology, which presents data in a similar format to that of the Wikipedia API, albeit in a 'cleaner' form (with human readable keys) as a result of the data been processed by hand. Although the keys are of a more human-readable format, some material is abstracted within the ontology and so is inaccessible with a basic query. An example of this is the page for 'London' (accessible at: `http://dbpedia.org/page/London`), which does not have a parameter `Mayor` with the value `Boris Johnson`. Instead it refers to a number of 'Leader Titles' and 'Leader Names', which include a reference to Boris Johnson

4.3.3 Choosing the Source and Interface for this Project

This project has a limited scope and time period available to it, and creating a system capable of identifying the relevant classes and then constructing SPARQL queries for looking up an abstract property is an example of an approach that is out of scope, as it involves natural language processing and semantic analysis of the names of the ontological classes, not just the query being made. As such, this project will make use of a combination of the data from the Wikipedia API and DBPedia top-level entity representations, collecting keys from both and working with the key with the highest semantic relatedness match to identify the data-value that most likely is the answer to a user's question. The DBPedia Ontology also provides access to the abstract at the same level as the infobox, so general descriptions of entities will be sourced from there.

4.4 Processing Natural Language Queries

Because of the nature of its audience, the system needs to be able to process input in natural language form rather than enforcing a formal grammar on users as described in User Requirement 8 and System Requirement 1. This involves constructing a natural language processing system to process queries and identify their meaning. In this project, queries are of a simple form: if a user is asking a question they are requesting a property of information about a given entity. An assumption is therefore made that each query consists of: a token representing the geographical entity been queried; a token representing the property been requested of that geographical entity; and some noise in the form of stop words.

Stop words are words that are frequently used in in language but do not carry any significance themselves [**?**]. It is common practise in Natural Language Processing to remove these stop words before attempting to process an input [**?**]. An example of a stop word is the word `the`.

It necessary to be able to identify the geographical entity in the query. This will be done using a Named Entity Extraction tool, of which there are many available online including Cortical [**?**] and Dandelion [**?**]. These tools extract keywords from a text and in some cases are capable of categorising keywords according to an ontology. For example, Dandelion links entities to the DBPedia Ontology [**?**].

The property about the geographical entity will be established by removing stop words, punctuation and the name of the geographical entity from the string. In the input `What is the height of the Eiffel Tower?`, this would result in the string `height`.

The result of this simple natural language processing should be that a sentence such as `What is the height of the Eiffel Tower?` is reduced to a structure indicating that the geographical entity is the `Eiffel Tower` and the property is the `height`.

4.5 Evaluating Success - Experimental evaluation of the system

Success will be evaluated by establishing if the project fulfils its original requirements, based on the scenarios. Many user requirements such as protecting the privacy of a user will be demonstrated through the software developed, whilst some that involve user interactions must be tested with users. As such, user testing will

be undertaken to evaluate three specific properties of the software through a trial usage and subsequent questionnaire: the usefulness and quality of the service, its usability, and the quality of the machine learning system.

Prior to running the experimental evaluation, a dry run will be undertaken with two participants. No data will be stored or collected - the purpose of the dry run is purely to establish early on any issues with the running of the experiment that would cause a problem during the real experiment. These participants will not be used in the final experiment as they will already have experience of using the software.

4.5.1 Client software for the experiment

The goal of this project has been to construct a piece of software that interfaces over SMS, as described in the project title. However there are practical limitations to running the experiment via SMS. During testing of the various services it was observed that HTTP-SMS Gateway Services added approximately a 5 second latency to both the sending and receiving of SMS, extending the length of time that a user would need to partake in the experiment to collect the same number of data points as could be collected when working straight over HTTP. In addition, the sending and receiving of SMS is costly and the savings made by not consuming SMS can be used to encourage participants to partake in the experiment.

The interface is also an important consideration for the experimental evaluation. A key goal is to collect as many data points from participants as possible, whilst still maintaining some of the context in which the software will be used - that of a mobile devices. With regard to input speed, it is assumed that typing speeds are slower on mobile keyboards than full-sized physical keyboards, so data points can be collected more rapidly when working on a full-sized physical keyboard. In order to do this, a mobile application will be developed that interfaces with the software, and this will be run on a laptop with a full-sized keyboard. This gives participants use of a physical keyboard, allowing them to query the system more quickly and therefore making it acceptable to ask them to perform a larger set of queries, whilst maintaining the mobile context.

4.5.2 Questionnaire Design

The questionnaire will aim to evaluate the usefulness of the system, the usability of the system and the quality of returned answers, and is based on the IBM "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information

Technology" questionnaire [**?**]. Questions were modified to suit the software being used in a situation without internet as opposed to a general workplace environment, and two questions were added: one to investigate the overall quality of answers returned to a user, and another to discover how trustworthy users felt the answers returned by the application were in general.

Each question is formed by statement and a measure of how much the participant agrees with the statement. This is measured by selecting a value on a five-level Likert scale [**?**]. The scale ranges from 1 through 5, where 1 is `strongly disagree` with the statement given, 3 is `neutral` and 5 is `strongly agree`.

A copy of the questionnaire that will be used is included in Appendix D.

4.5.3 Typical experiment plan

At the start of the experiment, participants will be given both the consent form, included in Appendix B and a document explaining what the software aims to achieve and what the participant should do, included in Appendix C. The consent form sets out exactly what participants will be asked to do, the information that will be collected from them, and reminds them that they can withdraw at any time. The instruction sheet sets out some topic areas for the participant to research using the software, with a varying level of freedom within that topic. For example, they may be asked specifically to `identify the height of the Eiffel Tower`, or to identify a property of an entity of their choice such as `identify the population of a country of your choice`. Participants will be asked to locate general information about a geographical entity. Constraints are placed on the area in which participants can query to limit the range of keywords used, thus focussing the machine learning onto a smaller group of words to allow a demonstrable improvement to be shown in the short space of this study.

After the experiment, participants will be asked to complete the questionnaire. Any additional comments arising during the post-experiment debriefing will also be noted beneath the questionnaire.

4.5.4 Assessing success quantitively

As stated in section 4.5, the experiment aims to evaluate three goals: the usefulness and quality of the service, its usability and the quality of the machine learning system. In the questionnaire, questions 1 to 6, 13 and 14 investigate the usefulness and quality of the system, whilst questions 7 to 12 investigate the usability of

the system. This project is heavily constrained with regard to time. As such the system's usefulness and usability will be deemed a success if it scores a significant positive average rating across all users in the questionnaire. The standard deviation of usefulness and usability ratings and the percentage of positive ratings will also be inspected to judge whether users had a consistent experience or if different users had extremely different views on the software.

The machine learning will be tested by inspecting the change in usefulness and quality rankings of the system made by users in the questionnaire as more participants use the system and provide ratings. The implementation of the machine learning system will be considered a success if a statistically significant positive trend is found in the usefulness and quality ratings when the ratings are sorted by the number of preceeding participants.

5 Design

In this section, the design of the system will be described, including information on: the structure of the server application and database; interactions with third-party services; systems used to protect user-identity; and the language, platform and libraries used for development. Diagrams will be given to describe parts of the application, and these will be in the ArchiMate 2.1 syntax [**?**].

5.1 System Design

In this section, the high-level design of the system software and its interactions with third-party services is described. The system design was driven by requirements listed in sections 4.2.2, 4.2.3 and 4.2.4, which were generalised to:

- The system must be modular with points to add additional plugins, for example for additional data sources.
- The system must be easy to maintain, with distinct boundaries between functional components and behaviour.
- The database must be local to decrease the risk of exposing data.
- The system must have clear interfaces for interacting with the external world.

Based majoritively from these requirements, the requirements in sections 4.2.2, 4.2.3 and 4.2.4, and the application of an informal gap-analysis between each iteration of the architecture, a modular system architecture was designed. This architecture details the modular structure of the application as well as some key pieces of functionality. A diagram showing the modular architecture is shown in Figure 5.1, and shows that the main application contains 3 core modules: a module for performing semantic comparisons of text; a module for Natural Language Processing (NLP); and a module for extracting and processing information from sources. This final module contains two further sub-modules, one for fetching and processing data from each of the DBPedia and Wikipedia data sources. Key application functionality from the top level of the application is also shown including

Query Parsing, where the type of the query is identified (for example, a question, a rating or a request for instructions) and Feedback Storage, where a feedback message is processed, linked to a previous question and stored in the database. In the NLP module the function for identifying the geographical entity been queried and the function for extracting the property a user is looking for from their question text are shown. Finally, in the Source Information Extractor and Processor module the function that applies past users feedback to answers to improve their quality is shown.

5.1.1 Micro-system for Protecting User Identity

In section 3.4.3, the need to protect the privacy of a user and the general technique that will be used (involving irreversible hashing) are described. The design of this system will now be discussed.

The main gateway into the system will be an HTTP call to a Python function called `sms()`, which takes the phone number of the user and the question being asked as its parameters. This function represents a core part of the privacy protection system, by generating the hashed phone number and only using that when calling the function that computes the answer. The user's unobscured phone number is only subsequently used when that function call has returned and it's time to send a reply SMS to the user. After that message has been sent the HTTP connection will close and the original number will be lost when the memory is released. As the computation of an answer only requires a general constant representation of user identity, not necessarily the user's real phone number, the specific value of the user's phone number is not required within the computation of the answer. As such, a hash is passed into the answer-computing function. This hash will be used within the database to store a user's history and profile. This means that only minimal user information (described in the physical data model in section 5.2.3) can be identified from the database, not the actual identity of a user.

5.2 Language, Platform, and Libraries

5.2.1 Language

The language Python has been selected to be the main language in project for four reasons:

1. Python is universally compatible and simple to execute. Once the server

application is written, it will be simple to migrate it between machines and adjust the application's configuration or environment.

2. Python is commonly used and has significant community support in the form of libraries for all the tools that will be used in this project, for example libraries for querying common databases and for sending text messages. This is useful as it will save time implementing common functionality like database calls.

3. Python is an extremely flexible language in which it is easy to implement applications of this type without syntax issues. This fulfils the system requirement in section (4.2.3, point 10) that the application is easy and quick to implement.

4. The author has significant expertise in the language.

A testing utility will also be created to query the application from a personal computer without using SMS. A simple OS X Cocoa application will be created that queries the HTTP interface with JSON in the expected format and displays an answer. Objective-C and and Mac OS X were chosen as the language and platform for this as the author has experience with Objective-C and uses a Mac. There was no requirement for universal compatibility because the application is only for testing and development purposes.

5.2.2 Platform

The development platform of the utility will be Linux. This is for three reasons:

1. Linux manages Python and its libraries in a simple way without complex installers with tools like the Python Package Index (a library repository) [**?**].

2. Common libraries that will be used for creating a HTTP server, making database queries and sending SMSs tend to be written primarily on the Linux platform.

3. The author has familiarity with Linux and has access to significant Linux infrastructure under a grant from Rackspace.

As previously stated, the interface to the application will be over the SMS platform.

5.2.3 Database

There are three tables of data that need to be stored. These are the tables for: user profiles, to allow the application to handle queries that make use of 'state' with users; a list of keyword matchings between the requested property in the question asked and the property returned from the data source with quality ratings from user feedback, which the application uses to learn from and improve its answers; and a record of all asked questions, their answers and the ratings provided, to allow for questions to be answered based on highly rated previous answers.

In the following section, the conceptual model is presented and then this is used to choose a database technology. Once the database technology has been chosen a logical data model and a physical database design are given.

Conceptual Data Model

There are four entities in the conceptual data model for this project. These are:

- **Users:** An entity that represents a user. The user is identified via an obscured representation of their phone number.
- **Last Question Asked by a User:** An entity containing the parameters of the last question. This allows the system to handle stateful queries that are based upon a previous query, such as when a user rates a previous answer.
- **Keyword Pair Quality Ratings:** An entity that contains the property requested by a user, the property returned to that user, and the number of times a matching was ranked as being either one, two, three, four or five stars.
- **Ranked Previous Answers:** A representation of a previously asked question, the answer text provided, and the rating given to the answer.

These entities are shown in the UML diagram in Figure 5.2, which also shows the entity relationships.

Database Technology

The database technology requirements for this project are:

1. The selected database must be as simple to implement and interface with as possible, whilst still supporting the system requirements. This is to allow for quick development and to avoid adding unnecessary complexity.

2. The selected database must allow for storage and retrieval of records with multiple parameters.
3. Support for records that are related to each other is not required, so relational databases are not needed, but not ruled out.
4. The database must have a library for interfacing with it in the language of this project.
5. The database should have built-in support for Sharding and Replication/Mirroring if possible to allow the application to scale. Sharding is the theory of distributing data and hence load across multiple servers [**?**], and replication is where data is stored on multiple machines to provide redundancy [**?**].

There are a number of database technologies available that would fulfil the database requirements of this project. Specifically, these can be split into two categories: relational databases and non-relational databases. Relational databases are used to store highly structured data in a table format [**?**]. Each table uses rows to represent a single record, and columns to represent the variables within that record [**?**]. These tables may contain constraints on the contained data to ensure integrity, such as limiting the range of a variable or enforcing that a variable may not be null [**?**]. Relational databases also encourage the use of table manipulation operations, where multiple tables are used to derive new tables with a processing stage in-between [**?**]. Relational databases also require a schema to be devised to formally structure the data [**?**].

Non-relational databases are specialised towards unstructured data, or where the data may be structured but there is a desire to not enforce a structure [**?**]. Records are stored in collections which are similar to a table in a relational database, and collections then contain documents, which are similar to rows in a relational database [**?**]. Unlike a relational database a collection does not have a set of columns or preset list of variables for each record [**?**]. Instead, each record is a dictionary. Non-relational databases generally support storing relational data (entities that refer to other entities) by nesting documents inside of each other for one-to-one and one-to-many relationships [**?**]. They also support referencing another document via its unique ID (which is assigned to all documents) [**?**]. A downside of this latter technique is that the first object must be retrieved from the database to retrieve the ID of the referenced object before the referenced object can then be fetched in a relational-like query.

The two common databases used for each of these database categories are MySQL databases [**?**] (relational) and MongoDB databases [**?**] (non-relational). In a Python application a MySQL database can be queried with an Object-relational mapping tool (ORM), which maps queries from the syntax of your programming

	MySQL	MongoDB
Fixed Schema	Yes, reducing flexible	No, giving flexibility
Querying Simplicity	Requires complex ORM and awareness of transactions, or construction of SQL queries.	Very simple with included MongoDB client.
Records support nested parameters	No	Yes
Supports relational queries natively	Yes	No
Built in Sharding support	Yes	Yes
Built in Mirroring support	Yes	Yes

Table 5.1: MySQL & SQLAlchemy vs MongoDB

language to a query in the SQL (a query language) and then maps the returned data to objects compatible with your programming language[1]. MongoDB comes with a simple pre-existing client to query the database natively [**?**].

To fulfil the database requirements, the database chosen should be flexible with no fixed schema, querying should be easy in the Python language, and either support for relational queries or nested documents will be required. Additionally, it would be good if the database natively has sharing and replication/mirroring support so that in future work a stable and reliable system can be built. MongoDB meets all of these criteria, with no fixed schema, a simple method for querying the database, support for nesting documents and sharding and mirroring support for future expandability. In contrast, MySQL forces a fixed schema for data and requires a complex ORM for querying, although it does support relational queries natively and has support for sharding and mirroring [**?**]. These properties of the two databases are shown in Table 5.1.

From this comparison, the decision to use MongoDB has been taken because MongoDB has more flexibility than SQL meaning that development will be less time consuming, and writing queries is easier in Python with the built-in MongoDB Library [**?**].

[1] ORMs also commonly apply optimisations to queries and validate them to ensure they are not malicious.

Table/Collection	Parameters
Users	(id, cellNumber, history, (lastQuestionText, lastQuestionRequestedProperty, lastQuestionReturnedProperty))
KeywordPairingRatings	(id, givenProperty, returnedProperty, oneStarRatings, twoStarRatings, threeStarRatings, fourStarRatings, fiveStarRatings)
RankedPreviousAnswers	(id, question, answer, rating)

Table 5.2: Logical Database Design. Underlined fields represent primary keys.

Logical Database Design

Having decided that the database technology will be MongoDB which supports nesting in section 5.2.3, the logical design of the database is able to reduce the number of entities from four in the conceptual model to three. This is because there is a one-to-one relationship between the `User` and `Last Question Asked by a User` entities, so the `Last Question Asked by a User` entity can be nested inside of the `User` entity as it only contains a small number of user-specific fields. This will simplify the implementation, meaning that a smaller number of collections will be needed to contain the entities.

The logical database requirements are listed in Table 5.2 and represented graphically in Figure 5.3.

Physical Database Design

As previously discussed in section 5.2.3, MongoDB separates groups or types of entity into collections (which are similar to a Table in an SQL database) but does not place restrictions on the structure of objects within a collection. The general structure of each document in a collection for this application is shown in Table 5.3. A small change was made between the logical and the physical database designs with the storage of ratings in KeywordPairingRatings. In the logical model these were individual counts of ratings at the five different levels (for example, a count of one-star ratings, a count of five-star ratings, etc) however in the physical model this was converted to a list of ratings received (e.g. `[5,5,3,4,5,3]`). This made the implementation more compact as five different variables did not need to be

Users	{
	'_id',
	'cellNumber',
	'history',
	'lastQuestion' : {
	'text',
	'givenProperty',
	'returnedProperty'
	}
	}

KeywordPairingRatings	{
	'_id',
	'givenProperty',
	'returnedProperty',
	'ratings'
	}

RankedPreviousAnswers	{
	'_id',
	'question',
	'answer',
	'rating'
	}

Table 5.3: Typical Document in each MongoDB Collection

addressed every time the ratings were read or written to.

The database software MongoDB will be hosted on the same server as the main application, as shown in Figure 5.4.

5.2.4 Libraries

In this section the libraries that were used in the server application in this project are presented. These include the libraries for querying the database, creating an HTTP server and sending SMS.

Database Querying

MongoDB comes with a Python client called PyMongo [**?**], which is developed by the same developers as MongoDB. It is the recommended tool to interact with a MongoDB database [**?**] and has good community support, so it will be used in this project.

Creating HTTP Server and REST API

The application needs to have an HTTP interface to allow interactions with it via SMS. As a result of the non-functional requirement "The system behind the SMS gateway must respond in a timely manner" in section 4.2.4, the application will need to be able to process multiple HTTP requests concurrently to prevent requests queueing up and responses being significantly delayed.

To create an HTTP Interface, the latest Web Server Gateway Interface (WSGI) specification (version 1.0.1, published as PEP 3333) [**?**] will be followed as it is the industry standard. The WSGI specification describes a general interface between web servers and web applications or frameworks (such as Twilio) in the Python language, thus describing how the web server should be constructed.

In this project, two tools are needed: a WSGI server, and a framework for writing WSGI applications. Flask will be used for the framework as the author has extensive experience using it and it is compliant with the WSGI standard [**?**].

With respect to the WSGI server, many implementations exist such as a simple WSGI server included in Flask [**?**] called Werkzeug [**?**], and the Gevent WSGI server [**?**]. Support for multiple concurrent HTTP connections is not a requirement in the WSGI specification, and so some WSGI servers such as the implementation of Werkzeug in Flask do not support this even though Werkzeug does natively support processing multiple HTTP requests concurrently. It does this by specifically enabling threading and increasing the default maximum number of processes from one [**?**]. The implementation of Werkzeug in Flask has these disabled in the source code because Flask uses thread-local storage (memory that is local to a single thread) and so cannot support concurrent HTTP requests on new threads [**?**].

A library called Gevent will be used to replace Werkzeug in Flask as it does support concurrent HTTP connections [**?**] using using the Greenlet library [**?**], which provides a notion of micro-threading (coroutines). When a new HTTP request is received, Gevent creates a new Greenlet [**?**], thus preventing the main thread being blocked and allowing multiple concurrent HTTP requests.

Sending SMS

There are a number of SMS providers with HTTP gateways available to choose from, all of which are similar. For this project, the provider must match the following criteria:

- Forwards received messages to a REST API to enable easy development.
- Supports sending messages via a REST API which also enables easy development.
- Has good coverage of countries so that users from many countries can use the service.
- Has a 'good' reputation. This is because user privacy is still an issue, since data is unencrypted when been sent over SMS, so the provider should be trustworthy.

It is difficult to measure the reputation of a provider accurately, but one possible way of doing this is by looking at the number of significant clients they have.

Two services were considered that matched these criteria. These were Twilio [**?**] and Plivo [**?**]. Both use REST APIs for handling incoming messages and for sending responses. Both Twilio and Plivo have comparable coverage as well: Twilio state that they send and receive SMS with numbers in 198 countries [**?**], and Plivo state that they support 207 countries [**?**]. The difference is trivial, as the number of recognised independent states in the world is between 190 and 200 depending on listing; the US Government recognises 195 [**?**].

With regard to reputation, for this project it was decided that Twilio is preferable to Plivo. Although it is difficult to measure reputation accurately, Twilio was founded in 2007 and has an group of clients including PayPal, Uber, Sprint and Airbnb, compared to Plivo, which was founded more recently in 2011.

Semantic Analysis - Measuring semantic relatedness between texts

Semantic comparison of text will be used for two purposes within the project: to match keywords from data sources, for example, if a user asks `how tall` something is, the property `height` would be appropriate to return; and to match questions with identical meaning to each other, so that good answers can be reused, saving the application from doing a full analysis of the data sources again. The requirements for these use-cases are different: for the former the tool needs to compare pairs of either single words of short strings or words for semantic

similiarity; and for the latter the tool needs to be able to compare full sentences for semantic similarity.

A description and comparison of semantic similarity and semantic relatedness is given in the Literature Review in section 3.3. Both use-cases in this application will be using semantic similarity as semantic relatedness is too general. An example using the datasource keyword matching usecase is that using semantic relatedness, the word `tall` would be related to `width` as well as the target word of `height` as a result of antonymy, potentially cluttering results. In the second use-case, consider a highly rated question in the database of `What is the population of England?` and then the question of `What is the population of London?` being processed. London is a part of England, so there is the potential for the population of England to be returned instead of the population of London.

A key requirement for the algorithm and method used to compute semantic similarity is that it can be implemented in compliance with the system requirement in section 4.2.3, point 10, that the system is easy to implement. A number of algorithms were discovered for calculating semantic similarity based on the WordNet lexical database [**?**], such as the Leacock-Chodorow and Resnik algorithms. Although mathematical representations of these algorithms are published [**? ?**], to fulfil the aforementioned requirement a pre-implemented semantic comparison tool is needed as implementation of these algorithms would be a non-trivial software engineering project. Both of the Leacock-Chodorow and Resnik algorithms are available in a Perl package as part of WordNet::Similarity, a set of Perl modules with an optional web interface for integration into applications. No other open-source libraries were found, so during application design, an attempt was made to use the WordNet::Similarity tools by calling the Perl scripts from Python code. This proved complex as there were numerous out of date dependencies to trace and install.

An alternative to using a specific algorithm in a library was to use an online semantics package. Two were identified: the Cortical Compare API [**?**]; and the Dandelion Similarity API [**?**]. Neither of these services document the algorithm that they use. Both take two pieces of text and return a value representing the semantic similarity of the texts. The Cortical API was found to be better suited for comparing single word or short multi-word terms using its `term` type [**?**], often returning errors when queried with two texts with a similar syntax, and in contrast the Dandelion API is better suited to single or multi-sentence blocks of text [**?**].

Both the Cortical and Dandelion APIs interface via a HTTP REST interface, so will be simple to implement. They are also in active development with support teams. For these reasons, they have been selected to be the semantic comparison tools used in this project.

Semantic Analysis - Discovering Named Entities in text

A named entity extraction tool is required to establish the geographical entity ('place') in a users' question. The requirements of the tool were:

1. Accepts a string of text and returns a list of keywords representing the place.
2. Keywords are categorised by a group classes, allowing for differentiation between a keyword representing a place, concept or person.
3. Accessible via a HTTP REST API for ease of implementation (fulfilling functional system requirement 10 in section 4.2.3).

During initial research, two keyword extraction APIs were discovered: the Cortical Keyword Extraction API [**?**] and the Dandelion Named Entity Extraction API [**?**]. However, the Cortical API does not fulfill requirement 2, simply returning a list of keywords. In comparison the Dandelion API returns a complex data structure which includes a list of DBPedia classes that the keyword is an instance of.

Research was done to identify a way of using the Cortical API to extract keywords and subsequently categorise the keywords after they have been been identified; however as there was no significant benefit of the Cortical API over the Dandelion API this was discounted and the Dandelion API chosen.

Results from the Dandelion API were checked against the `Place` class in the DBPedia Ontology by checking their `types` list (a list of DBPedia classes that the keyword fits into) for the string `http://dbpedia.org/ontology/Place`.

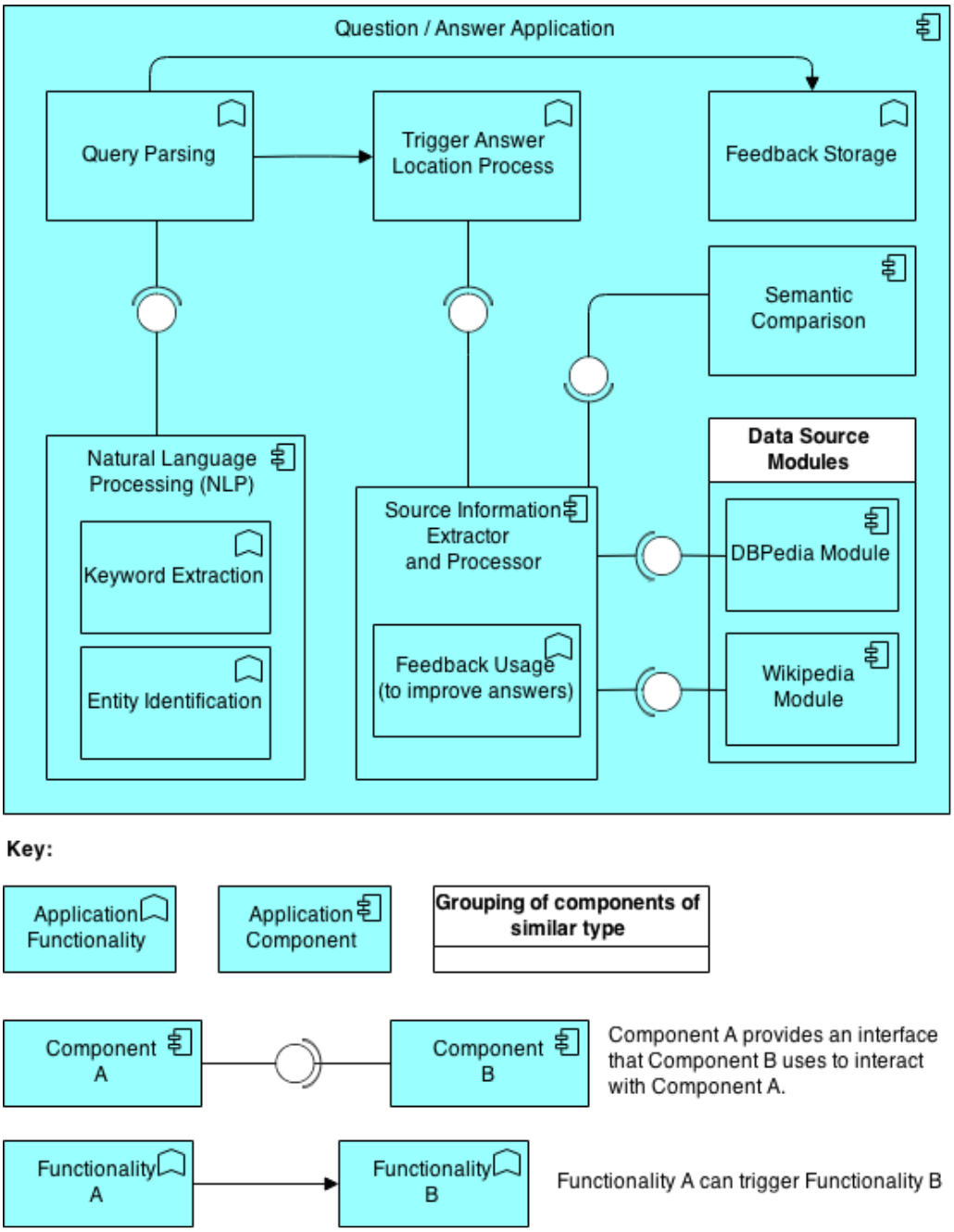

Figure 5.1: Component View of the system in ArchiMate notation [**?**] showing the modular structure of the application. The diagram shows the modules used when computing an answer to a question and the interfaces between them.

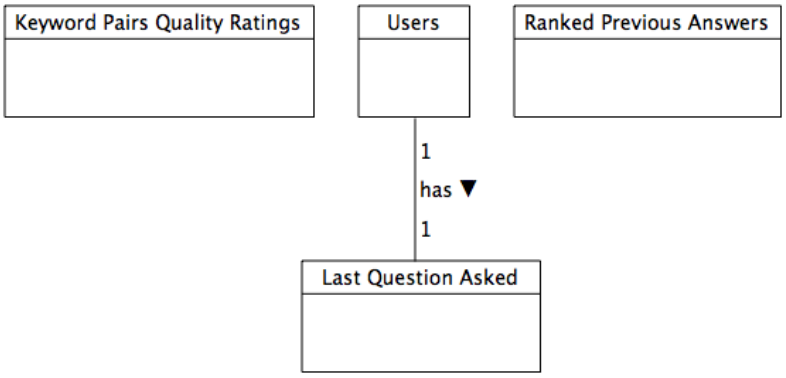

Figure 5.2: Conceptual Model of Data

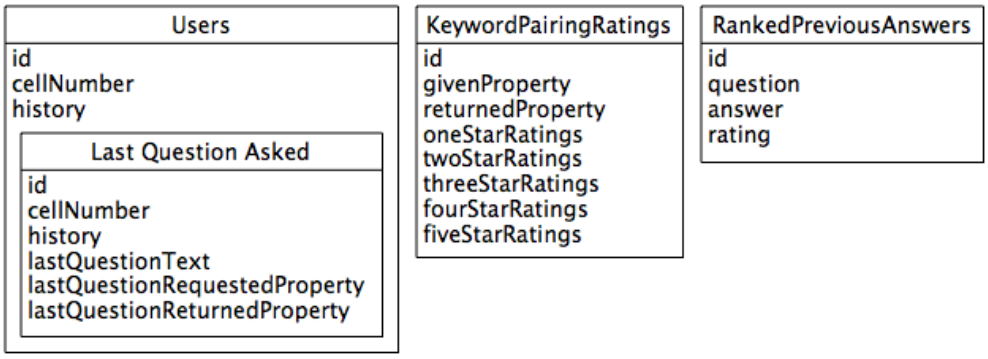

Figure 5.3: Logical Model of Data

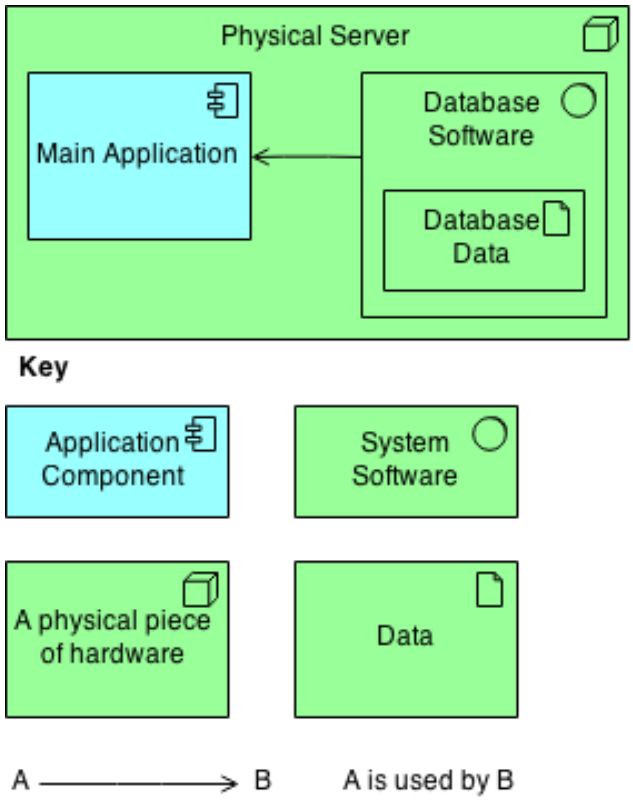

Figure 5.4: Deployment View of the system in ArchiMate notation [**?**]. The diagram shows the structure of the main server, which both runs the application and hosts the database.

6 Implementation

This section will describe the implementation of the server application and the clients developed for testing and evaluation. In the server application, focus will be given to the tools and services used, the natural language processing system, the method used to identify answers and the machine learning aspects of the application.

Code used within this project and discussed in this section is located in the following Git repositories:

- `https://github.com/samheather/dissertationServer` - This repository contains the code for the server application.
- `https://github.com/samheather/dissertationClient` - This repository contains the Xcode Project and code for the Mac OS X testing client, used in development of the application.
- `https://github.com/samheather/dissertationIos` - This repository contains the Xcode Project and code for the iOS client application, used in the experimental evaluation of the application. It has been constructed for an iPhone 5 screen size.

6.1 Implementation Overview

As discussed in section 3.5.5, the implementation was completed whilst following the iterative development process. Software development took places in stages which represented each iteration, and after each iteration a version of the software was demonstrated in a weekly meeting. In these meetings, the software's capabilities and success rate were discussed, and requirements were re-evaluated based on progress. Ideas for expansions to the software (for example, the Natural Language Processing, which was not in the original plan) were also discussed.

The software has been constructed in a modular manner to allow easy future expansion and modification. This modular structure is described in section 5.1, where the system design is presented.

6.2 Tools and services used

The following tools and services were used in the implementation. These were chosen to fulfil the requirements detailed in section 4.

- Python was the programming language used in this project. This was due to a number of reasons, including that the author had experience with writing an application with some similar characteristics as those in this project.
- MongoDB version 2.6.6 was used as the database software in this project.
- Both the application server and the database server were hosted on an Ubuntu version 14.10 "2 GB General Purpose v1" Linux server, hosted by Rackspace in London.
- No integrated development environment (IDE) was used in the development of the server application. The author chose to use textural editors due to familiarity with them, and it was felt that the assistance provided by an IDE with managing modularity/abstraction/semantic checking would not be of significant benefit, bearing in mind the size of the server application.
- An IDE was used in the development of the client applicatios to reduce the use of expensive SMS during testing. This was written in Objective-C for Mac and iOS, and created in the Xcode IDE, which is the industry standard.
- The Twilio SMS Service [**?**] was used to facilitate the SMS interface to the server application.
- The Cortical [**?**] and Dandelion [**? ?**] APIs were used for semantic analysis of text.

In addition, the following Python libraries from the Python Package Index [**?**] were used within the server application:

1. Greenlet [**?**] - provides a notion of micro-threading (coroutines), which allows the server to process multiple HTTP requests concurrently.
2. gevent [**?**] - a coroutine based networking library for Python. gevent provides a Web Server Gateway Interface (WSGI) server that supports processing multiple requests concurrently, using Greenlet.
3. PyMongo [**?**] - Python library containing tools to work with MongoDB databases.
4. ujson [**?**] - Twilio uses JSON as the format for HTTP requests, so it makes sense to use it for all requests. ujson allows for effecient encoding and

decoding of JSON to and from Python data structures.

5. Twilio [**?**] - Twilio library used for receiving and sending SMS.
6. lxml [**?**] - Used in the 'Wikipedia Utils' library, which handles interactions with the Wikipedia API. lxml is used for processing XML/HTML in Python.
7. Flask [**?**] - a micro-framework for web applications, used for its simple syntax for defining HTTP routes (URLs that trigger Python functions). Flask also comes with a WSGI server, but it does not support processing multiple HTTP requests concurrently, so it is replaced with the Gevent WSGI server in point 2.
8. NumPy [**?**] - Numerical Python (NumPy) is used for outlier detection in the rating system. It provides a function call for calculating the standard deviation of a set of numbers.
9. re - a Regular Expression matching library built into Python which is used in the 'Wikipedia Utils' library and in application code for converting Camel-Case text to Snake case text.
10. Wikipedia Utils [**?**] - a library used for abstracting the Wikipedia API.

6.3 Implementation of SMS interface

The SMS interface to the application was developed using the Twilio SMS service and Library [**? ?**]. A HTTP interface to the application was hosted using Flask [**?**] at `<server_url>/sms`, and the Twilio service was configured to make a request to this URL when an SMS was received. The function `sms()` is executed upon receiving a request on the `<server_url>/sms` address, which extracts the phone number of the user and the text of their message from the request. The number is obscured, as explained in section 5.1.1, and then the function `start()` is called with both the obscured number and the text from the message. This function then computes the answer to return to the user, described in section 6.5, and the response is sent back to the user using the Twilio library to make a HTTP call to the Twilio service.

The service provided by the application and its service are shown in the Archi-Mate diagram in Figure 6.1.

6.4 Implementation of natural language processing module

In this section, a description of how the natural language processing module was implemented is given.

It was decided that a module should be constructed specifically for the parsing of natural language questions in the application. This module, called nlp, processes question in natural language form, and is able to output the two parameters needed by the application for answering a question.

The first step in the process is to remove punctuation from the question text, which is of no meaning in the context of this application. Consider an example sentence `How tall is the Eiffel Tower?`. In this sentence, the question mark will be removed to give `How tall is the Eiffel Tower`.

The next step in the process is to identify the geographical entity that a user is investigating. This is done by passing the original question into a Named Keyword Extraction tool, which extracts a list of keywords representing entities from a string and returns them. The Dandelion Named Entity Extraction tool was selected as in addition to providing a list of words representing entities, it categorised them according to the DBPedia Ontology. By using this, it was possible to identify the geographical entity by simply checking each keyword for the type `http://dbpedia.org/ontology/Place` in the types associated with each keyword. This is done by the `extractEntities()` function in the nlp module.

The final step in the process is to identify the property that a user is requesting about the entity, for example, the height. This is done in the `stripToProperty()` function in the nlp module. This function identifies the property by assuming it will be the only text in the sentence with meaning, excluding the place name. As such, the first action of this function is to remove words relating to the name of the geographical entity being queried, which it does by checking each word in the question text against each word in the place string, and removing the word if they are equal. After this process, our example question will have been changed from `How tall is the Eiffel Tower` to `How tall is the?`. The next step is to remove meaningless words that do not identify the property that the user querying. These are stop words, such as `and, the, of`. As with removing words relating to the geographical entity, this is done by checking each word in the question text against a list of stop words and removing them if they match. In our above example, the question text will now be `tall`. This is the property determined to be the subject of a user's query.

Having called both `extractEntities()` and `stripToProperty()`, the NLP module packages these results into a dictionary and returns them as the processed representation of the question text.

6.5 Implementation of answer location system

In this section, a description of the process followed to answer a question about a property of a geographical entity is presented.

After having identified the place and the property of that place in the question text using the NLP module, the application must locate the value of that property for that place. This is done in the `sourceProcessor` module, which collates data from the data sources and then attempts to identify from those sources an answer to the user's question.

The entry point into `sourceProcessor` is the `findArgumentOnPage()` function which takes the place name and the property being queried as input parameters. The first step of `findArgumentOnPage()` is to query the data sources for a dictionary of properties about the geographical entity, pulling down all information available on the entity in question. Interfaces to data sources are factored out into modules, such as `dbpediaParser` and `wikiPageParser`, which handle queries to these data sources and return the data to the application in a normalised form (a dictionary of keys and values about the geographical entity). The modules controlling the data sources have a common interface: a function `getInfobox()` which returns a dictionary of keys and values from a source. Additional data sources can easily be added to the application by conforming to this interface.

After having retrieved a list of properties about the entity, the application then iterates through the datasets (one dataset per source) checking the keys in the dictionary against the requested property in the question text. If any key in the keys for a dataset is an exact, case-insensitive match to the requested property, the value for that key is returned as the answer. Otherwise, each key in the dataset is compared to the requested property text for semantic relatedness. This tries to match key strings given in different contexts and is useful in the case that a user asks a question using a different word to represent the property they require than the one in the data source. After having compared each key to the property given by the user, the key with the highest semantic similarity is returned.

An example of where the semantic comparison and matching of keys is useful is the question `How tall is the Eiffel tower?`, where the property string

is `tall`, and in the data source the key might appear as `height`. The word `tall` has a high semantic relatedness to the word `height`, so the value for `height` will most likely be returned, unless another key with higher semantic similarity is found.

This semantic comparison is done by the the `compare` module, which has a function `similarityOfProperty()`. This function takes two strings as an input, and returns a value representing the semantic similarity of the two strings. The Cortical API [**?**] is used for the semantic comparison.

The Cortical API is not configured for recognising text in CamelCase, so helper functions such as `camelCaseToSnakeCase` and `camelCaseToSpace` are also included for converting CamelCase to either Snake Case or space-separated words.

A minimum threshold was set for the semantic similarity of keys that could be returned to prevent a key with a very low semantic similarity been returned as the answer where the data source didn't have the requested information available. This helps to prevent incorrect answers been returned to users. The Cortical API documentation recommended a threshold value of 0.3 [**?**], however this was resulting in some key matchings been ignored. Contact was made with the Cortical team who suggested a value of 0.2 might be more appropriate for this use case, and during development all key pairs and their semantic similarity values were printed so that when an answer was not found the semantic similarity value of the correct matching could be inspected with the potential to adjust the threshold. The advice from Cortical turned out to be correct, with numerous examples of requested keys having a similarity score to those from the data sources in the range `0.2` through `0.3` during development, such as the requested key `population` and the key `population urban` from the DBPedia dataset having a similarity score of `0.24695`.

Initially it was planned to compare questions to historical highly rated questions with extremely high semantic similarity. If the database contains the highly rated question `What's the height of the Eiffel Tower?` and then the new question of `How tall is the Eiffel Tower?` is asked, the answer that was provided for the first question would be sent for the second question. Unfortunately there were issues with implementing this such as how to efficiently check a new question against every historical highly rated question for semantic similarity. Although it would be possible to implement this by storing a list of keywords for each question and then only semantically comparing questions that share keywords to reduce the number of questions that are compared, or by storing semantic representations locally and therefore doing the comparison locally, this would have taken too much time. As a compromise, the current version of the

software does collect question texts and their ratings to begin building a database. In the future when this feature is implemented, this historical database will be useful as it means there will already be data for the application to use rather than having to wait to build up a dataset of full question texts and ratings.

The comparison of question texts is also done in the `compare` module which contains a function `similarityOfQuestion()`. This function uses the Dandelion [**?**] similarity API to compute the semantic similarity of two longer pieces of text such as questions to the application.

6.6 Implementation of rating and machine learning system

The application developed uses user ratings of answers to questions to learn about the quality of answers and to improve future answers. The implementation of this can be split into two sections: the collection of ratings and the use of ratings when answering questions. A five 'star' scale is used for ratings: a rating of one indicates that an answer was of low quality and a rating of five indicates that an answer was of high quality.

The machine learning system built uses Machine Learning from the Reinforcement Learning paradigm, as described in the Literature Review in section 3.2, as this suits the concept of users providing feedback which only indicates if an answer was of high or low quality. The general technique used is that of Learning by Reward/Punishment, also described in section 3.2, where ratings in the database are used as weightings to adjust the parameters in the answer location process.

6.6.1 Collecting Ratings

When a user asks a question and an answer is found the function `updateUserWithLastQuestion()` is called. This stores the text of the question, the requested property interpreted from the question text, the geographical entity name and the name of the property returned from the data source in an object called `lastQuestion` inside the `user` object. This means a record of the last question a user asked is always kept by system, allowing them to send another message to the system that relates to their previous question.

This is used by the ratings system. When a message is received of the form `rate X` (where X is an integer and in the inclusive range of 1 and 5) the system

can identify what the last question was and what parameters it should be applying the rating to.

Two ratings are stored by the system: the `Keyword Pair Quality Ratings` and `Ranked Previous Answers`. The `Keyword Pair Quality Ratings` is a measure of the quality of a semantic matching between the desired key in the user's question and the key returned from the data source. In the case that the answer returned to a user is unrelated to what they asked, for example they asked about the height of a building and were given the name of the city it is in, the wrong key from the data source will have been returned. After receiving a low rating, the application learns that the key requested by the user does not map to the key that was originally been returned as the answer (in this example, it can learn that the key `height` does not semantically match a key called `city_name`).

The second set of ratings, `Ranked Previous Answers`, are only stored within the database and are not used to answer questions. This is due to time constraints within the project as implementing this would have taken a significant amount of time. However it was decided that even though it was not possible to provide answers based on `Ranked Previous Answers` the collection should still be constructed so that in future work a pre-existing dataset already exists to use rather than having to wait to collect more user ratings. An explanation of how this would be used is provided at the end of section 6.5.

Ratings are stored using the function `adjustRanking()`, which is called with the the following parameters from the question been rated (the last question): the full question text; the answer given; the property quested in the question of an entity; the property sent to the user from the data source; and finally the actual rating number. The `adjustRanking()` function first inserts a record into the `Ranked Previous Answers` collection containing the question text, the answer given and the rating the user gave. Next, the function checks if any rating records already exist in the `Keyword Pair Quality Ratings` collection for the requested and the given keyword. If they do the document is retrieved, or if they don't a new document is created. In that record, the rating is added to the `ratings` list, and the document is updated/saved back into the collection.

6.6.2 Using ratings to improve answer quality

Rating are used in the `sourceProcessor` module to provide answers of a higher quality to users. When the similarity score has been computed for a pair of properties using the `compare` module the func-

tion `adjustSimilarityWithRanking()` is called. This takes the similarity score that was returned, the property currently being inspected in the data source and the property that the user requested. The `adjustSimilarityWithRanking()` function then does a lookup in the `wordReferencePairs` collection using the two properties to identify if any ratings exist between the requested keyword and the keyword found in the data source. If no ratings do exist the similarity score is returned unchanged. If there are ratings, adjustments are made to the similarity score to take into account whether the property in the data source has been rated as providing a good answer or a bad answer in the past. This is done by applying a multiplier to the similarity score for each rating.

The multipliers selected were driven by the goal to see an improvement during the experimental evaluation, such that a single answer would be replaced by another answer within the course of the experiment. The following values were used in the implementation:

- For each rating of `1`: reduce the semantic similarity by 5% (multiply by 0.95)
- For each rating of `2`: reduce the semantic similarity by 2% (multiply by 0.98)
- For each rating of `3`: do not change the semantic similarity
- For each rating of `4`: increase the semantic similarity by 2% (multiply by 1.02)
- For each rating of `5`: increase the semantic similarity by 5% (multiply by 1.05)

These values were found using trial and adjustment but were tested with two particular use cases with the intent that an improvement would be seen in the course of the experiment:

In the first case, when `population` is requested, the value for `areaUrban` is returned with semantic similarity of 0.30488 (5 significant figures) instead of `population_urban`, which also has a semantic similarity of 0.30488.

In the second case, when the property `height` is requested of Mount Everest (in the context of `what is the height of...`), the value for `width` is returned with semantic similarity of 0.34451 instead of `elevation`, which has a semantic similarity 0.24695.

In the first case only one rating is needed for the application to begin sending the correct property, however in the second case seven ratings of one 'star' are required before the application starts returning the elevation of Mount Everest as opposed to its width.

The values for the adjustments for the ratings were chosen with the aim of preventing dramatic 'swings' or changes in the application behaviour caused by a small number of ratings. A disadvantage of the current adjustment values is that it can take a number of ratings for the application to learn the correct answer, such as in the case of the `height of Mount Everest`. However it was observed during development that typical similarity scores between a requested property and all the keys in a data source typically varied from 0.1 through 0.4. If the adjustments were to be larger, for example 15% for a rating of `1` or `5`, a typical similarity score of 0.25 (in the center of the common score range) would have been adjusted to 0.437 after just three positive ratings of `5`. This is outside of the common range, thus it will quickly dominate any new keys that are added to the data source which could be more relevant.

A potential issue with the ratings system is that of handling incorrect and anomalous ratings, whether caused by user error or malicious attempt to damage the system. If the matching of pair of keys has the ratings `[4,5,1,4]` it appears that the rating of `1` is anomalous and should be ignored from the adjustment process. This is done through the use of the standard deviation of the group, which is calculated in the function `removeOutliers()`. The standard deviation of the group of ratings is computed to calculate the spread of the majority of results. Techniques exist for choosing the multiple of the standard deviation from the mean that should be used for finding outliers by choosing a desired significance level, such as Grubbs' test [**?**]. However, as the range was limited and small with only five possible values it was decided that to keep the implementation simple and conserve time that a single multiple of the standard deviation would be used constantly.

The multiple of `1` was chosen for the standard deviation based on numeric experimentation meaning that results that are outside of the range of one standard deviation from the mean are then excluded from the adjustment process. Where the standard deviation from the mean is a decimal, it is rounded to its nearest integer to establish the range of ratings that will be included in the computation. The value of one standard deviation was selected as the range of ratings was so small (with only five possible values for a rating) that a larger multiple of the standard deviation made it likely to include anomalous data. This is shown with the above example data, where the mean is 3.5 and the standard deviation is 1.5, meaning that the range of valid ratings is 2 through 5 inclusive which successfully excludes the outlying result. If the multiple for standard deviation was increased to 1.5 or even the commonly used value of 2 the range from the mean would increase to 2.25 or 3.0 respectively, meaning that the range of valid ratings would be 1 (rounded from 1.25 for a standard deviation multiplier of 1.5) through 5, which includes the incorrect value.

Using this method there are situations where a result which is perhaps not an outlier (depending on the method used to categorise it) is excluded. An example of this is if a pair of keys has the ratings `[4,4,4,4,4,4,5]`. It's reasonable to expect that the there is some variation in the opinion of users as to whether the answer provided is excellent and so scores `5` or just good with a score of `4`. However in this situation the `5` is excluded using the aforementioned standard deviation of `1`. In this group of ratings the mean is `4.14` and the standard deviation is `0.35`, which will mean the included range of valid ratings is only those with value `4`.

A possible solution to this would be to ensure that ratings within one star rating of the mean are always included and not classified as outliers. For example if the mean of a list of ratings is `3.14` then, regardless of the aforementioned outlier detection techniques, ratings of `2` and `4` would not be treated as outliers. This was not implemented for the experimental evaluation as it was an enhancement that was thought about in the later stages of the project.

6.7 Implementation of Testing Client

To aid development of the server application, a desktop client application was developed which interacted with the HTTP interface of the server application to allow queries to be tested quickly as the software was developed. The application, shown in Figure 6.2, allows both the user's phone number and the question parameters to be set and sent to the server. The raw returned output is then displayed regardless of whether this was an answer or an error, thus making testing and debugging both quicker and easier.

The application was built in Objective-C using the Xcode IDE, as described in section 6.2. The application used the phone number and question text variables to construct a HTTP query which was then sent to the server. The responses was then displayed in the application. The simplicity of this application makes it easy to port to other platforms, as was done in section 6.8.

6.8 Implementation and iteration of iOS app used in experimental evaluation

As described in section 4.5 an iOS app was created to act as a query interface for the server application for the experimental evaluation. This was written in Objective-C

meaning it was trivial to port the implementation used in the testing client in section 6.7.

After running the dry run of the experiment three usability issues were identified resulting from the use of the iOS Simulator on the Mac platform: the on-screen keyboard remained visible covering text on some longer answers; the participant ID field at the top of the screen caused confusion with one dry run participant attempting to change the value; it was slow to clear the question field to send a new query. Also, both users would press `return` as opposed to clicking the `Go` button, which wouldn't send a question.

Fixes were made to the software before running the full experimental evaluation such as:

- Questions are now sent both if either the `Go` button is tapped or the `return` button is pressed on the keyboard.
- When a query is sent, the on-screen keyboard is hidden as seen in Figure 6.3. It can be brought back by tapping on the question field.
- When selecting the question field, all the text is selected by default making it easier to type a new query without spending time deleting the old text. A 'clear field' cross button which is standard on the iOS platform is also shown. This is shown in Figure 6.4.
- The field to enter participant ID is hidden, as shown in 6.3. This is now requested in a modal alert when the application starts: the ID is then entered by the researcher.

6.9 Adding Multi-Language Support

As described in section 4.1 the software was initially intended to have support for multiple languages. Although this was not included in the software developed, the software has been constructed with an awareness of this potential future feature so that it can easily be added.

On a high level there are two potential methods of implementing multiple language support. The first method is simply to translate questions into the language of the system when the question arrives and then to re-translate the answer to the language of the user before sending it to them. A key advantage of this is that very little of the implementation will change, however as the application will still be working with English resources like the English edition of Wikipedia and then translating back it's likely that the quality of the answers will not be as high as if

the application were to work natively in multiple languages.

The second method is to adjust the application to work natively in multiple languages. This involves adjusting the `Natural Language Processing (NLP)` module described in section 6.4 and the tools used to make semantic comparison so that comparisons can be made of words in non-English languages. The `NLP` module currently has two pieces of functionality specific to the English language: the identification of geographical entities in a question text (entity extraction) and the removal of stop words, described in section 6.4. With regard to entity extraction, the Dandelion Entity Extraction API used for identifying geographical entities has built in support for five languages, where the language can be set to an Auto-detect mode or to a specific language via a parameter in the API call [**?**]. The second piece of functionality in the `NLP` module, that of removing stop words to obtain the property string, is dependent on having a list of stop words in the language of the question. These would have to be collected from various sources and language identification would need to be used to identify the correct set of stop words to use for a question. Finally the `compare` module would need changed to support comparing words in non-English languages. Currently this module uses the Cortical API [**?**] which currently only supports the English language [**?**].

Due to the second method requiring substantial engineering work to support the removal of stop words in the NLP module, the requirement to find new tools for semantically comparing words, and the entity extraction only supporting five languages, the first method will be used when multiple language support is added. This will be done through the addition of a new `translate` module into the system. This module's interface will be two functions: `translateQuestion()` for translating an input question to English and `translateAnswer()` for translating an answer back to the language of the user from English. The `translateQuestion()` function will return a dictionary containing a representation of the original language that the question was in and the translated question in English. When translating the answer back to the language of the user with the `translateAnswer()` function, the function will take the original language as a parameter to indicate which language it should translate the answer to. The `translate` module will use an online language detection and translation API such as Google Translate [**?**] to both detect the language used in a question and translate text between languages.

The module will be implemented into the application between the SMS interface and the `start()` function, described in section 6.3. The question will be translated before the `start()` function is called, the original language will be stored, and then the answer returned from the `start()` function will be translated back into

the user's language.

6.10 Discussion of implementation and issues faced

Software was developed following the plan based on the iterative model, which was described in section 4.1. In this section some of the limitations and issues faced with the implementation will be described and methods for expanding the modular software will be presented.

6.10.1 Issues faced in the implementation

One of the key issues that was faced during the implementation was that the APIs provided by Cortical for semantically comparing text were still been heavily developed as Cortical evolved their product. These changes were not documented so caused issues when they happened. One such example is when the API started rejecting some strings with space characters in them that it previously had accepted. This was fixed by replacing space characters with underscore characters when checking keys in the data sources for semantic similarity. Another example is that the API started returning HTTP errors for some specific comparisons instead of ratings of zero as it had previously. This occurred when the keys for the input parameters to the HTTP query were incorrect as the API is unclear on when to classify an input as a `term` or a `text` [**?**]. Again this was undocumented and required an unexpected change to the application during development.

Another issue faced was that the data sources did not always present the data in an ideal format. One data source where this occurred is the Wikipedia API, which would inconsistently return measurements as either a number without a unit or a string containing the data in different units. An example of this is a query for the property `height` on the page for the Leaning Tower of Pisa, which returns the string `convert|55.86|m|ft`. Basic efforts were made to resolve this by removing unusual symbols such as the vertical bar (pipe) but it was not possible to identify and remove incorrect units and words. In the future it may a consideration to remove Wikipedia as a data source if other 'cleaner' data sources can be found. An ideal data source uses simple and descriptive names for keys in the data source to allow the semantic analysis to link the terms from a users query to the key names, and the values in the data source are only the data been measured, without additional words and symbols as the Wikipedia API provides.

The structure of the DBPedia data source caused another issue. This was that

some properties of geographical entities remain inaccessible to the application because they are abstracted into linked objects in the DBPedia Ontology. An example of this is the object `London` which does not have a field `Mayor` but instead a pointer to another object called `Mayor_of_London` which contains the name of the Mayor of London. An explanation of why data from these objects can not be retrieved is given in section 4.3.2 and a suggestion for adding this as a feature in the future is presented in section 9.6.

A known limitation of the implementation is that the type of questions that the application would handle are also limited to questions which ask about a specific property of a known geographical entity, or for a description of such an entity. The application can not handle complex questions that are not of this form, such as `What is the depth of the deepest ocean?`. This does restrict the type of questions that can be asked of the application. To extend the application to work with these more complex questions would require substantial work and it was decided that this was out of the scope of this project.

6.10.2 Extending the Application

The application has been constructed in a modular manner to provide good maintainability. Code that handles a specific and distinct piece of functionality is abstracted into a separate module, such as Natural Language Processing functionality which is in a module called `NLP`, and the code that is used for doing semantic comparisons which is in a module called `compare`. This is shown in the Component View diagram of the application in figure 5.1 in section 5.1, where the diagram is also described. A key advantage of this modulear design is that modules can be swapped in and out and external services change. For example, if the semantic comparison service currently used by the application becomes unavailable, the `compare` module can be changed to use another service without affecting the rest of the application.

The application was also designed to be easily extendable. One example of this is the simple process to add additional data sources to the application. This is done by creating a module for that data source that conforms to the interface described in section 6.5. The interface dictates that the module needs to have a function `getInfobox()` which returns a dictionary of the keys and values on a given page. That module is then simply added to a list of data sources in the `sourceProcessor` module and then the application will use that data source when computing answers.

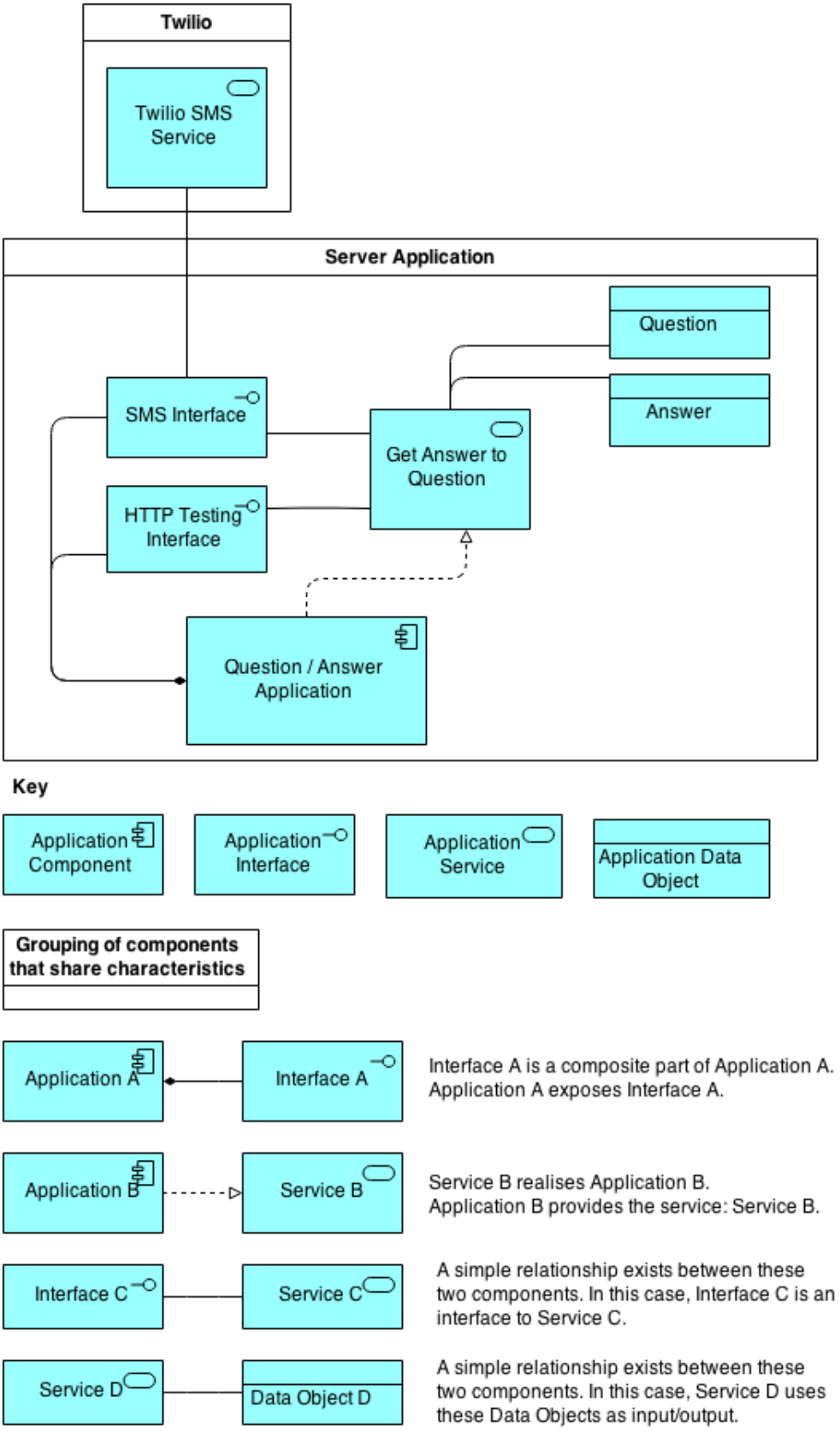

Figure 6.1: Application Behaviour View of the system in ArchiMate notation [**?**]. This diagram shows the question/answer service provided by the application and the SMS interface to it. The HTTP interface used for development and testing is also shown for completeness. [**?**]

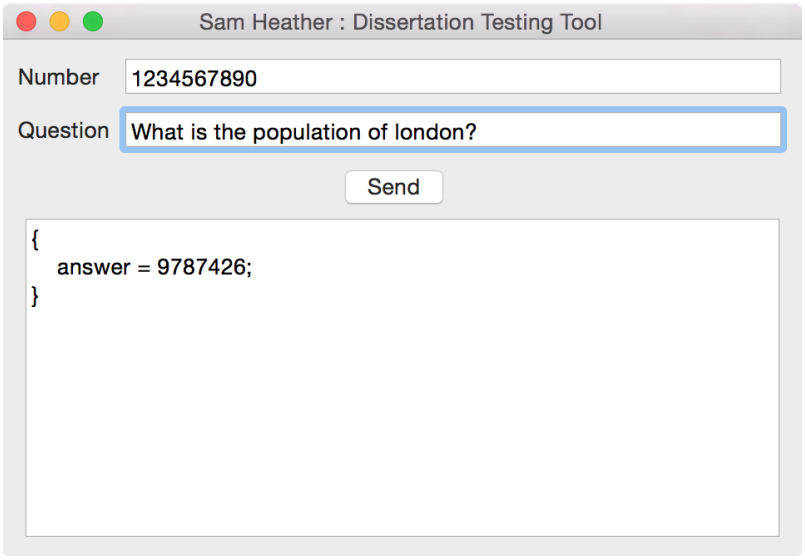

Figure 6.2: Development and testing client for Mac OS X showing a sample query

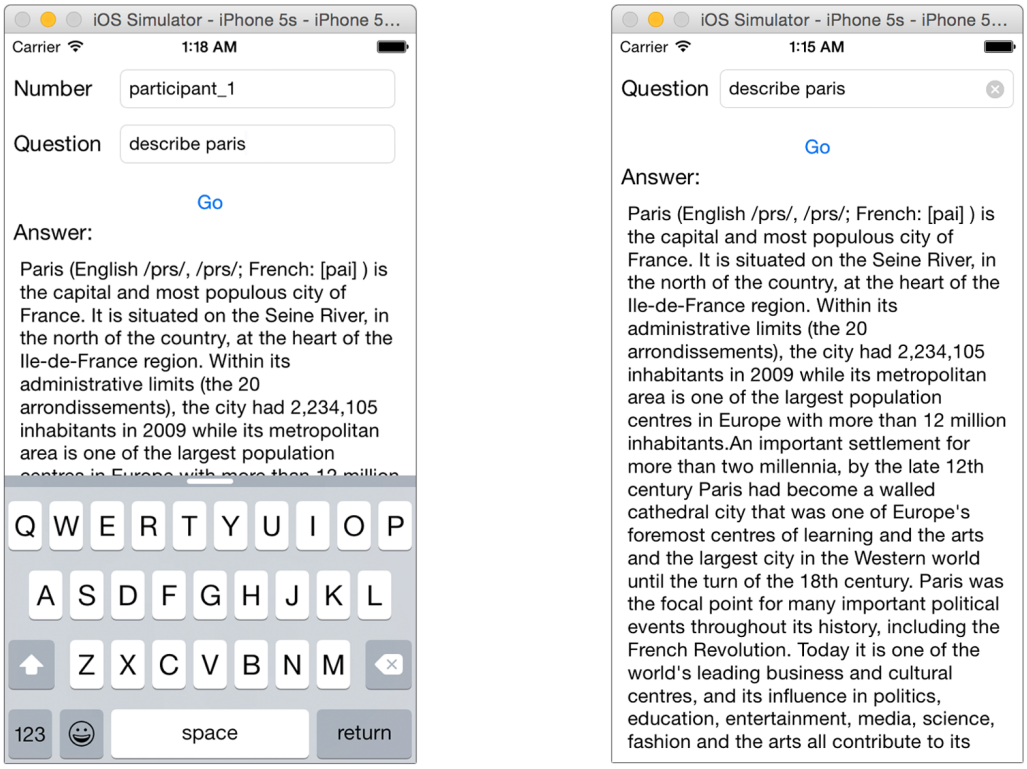

Figure 6.3: Left: Original application. Right: Revised application with keyboard hiding.

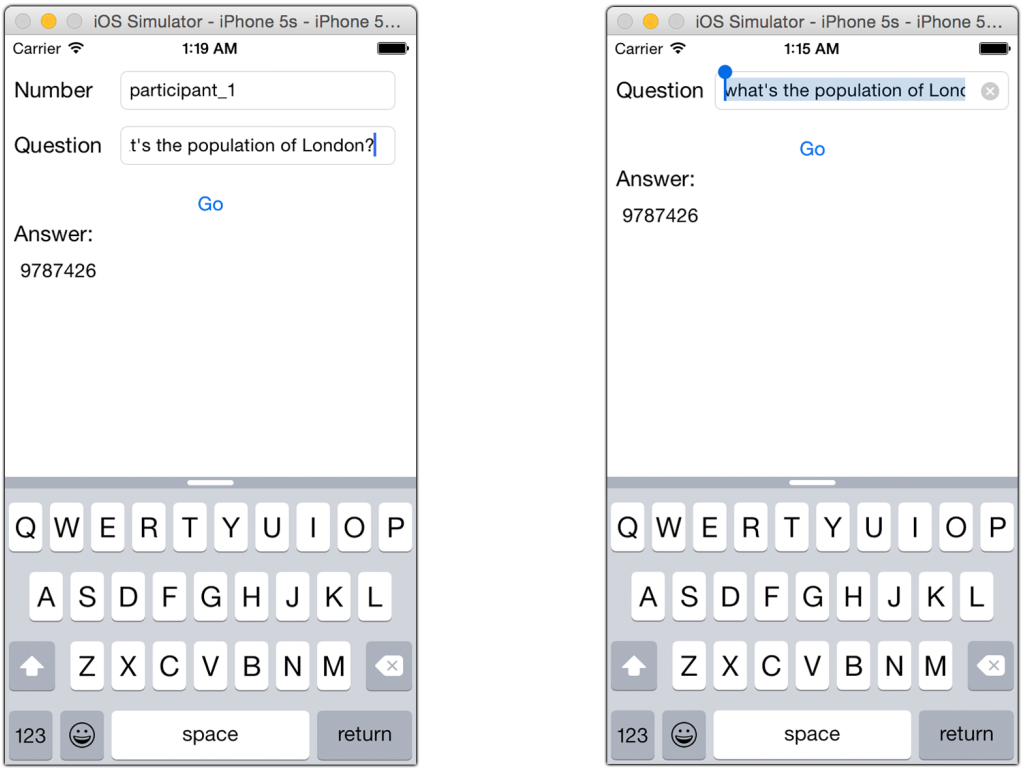

Figure 6.4: Left: Original application. Right: Revised application with hidden participant ID field and Question field with a clear button and auto-selection of text when it's tapped to bring the keyboard up.

7 Results and Discussion

In this chapter the results from the experiment will be presented, analysed and discussed, and information on the pool of participants will also be provided. The key piece of research been investigated is whether the application learns effectively, although usefulness and usability ratings will also be looked at.

7.1 Participants

21 participants were sourced to take part in the experimental evaluation of this project. Participants were mainly sourced from the University of York however a small number did come from other locations such as Bristol when the researcher made an unrelated trip there.

Participants were heavily skewed towards young students: 17 participants identified as being aged 18-24 and 14 as students. Of the remaining participants, two were aged in the range 25-34 and the remaining two in the range 45-54.

Three of the non-student participants identified as been self-employed, one identified as been employed and one as retired. Additionally, two participants identified multiple occupations: one selected employed and self-employed, whilst another indicated self-employed and student.

All participants in the experiment were regular internet users with all participants identifying their typical Internet usage being 'multiple times per day' or greater. Seven participants selected 'multiple times per day' whilst 14 selected 'more than once per hour'.

7.2 Results

In this section the results collected from the experiment for application usefulness, usability, and whether the application learns effectively, will be presented, analysed and discussed.

7.2.1 Usefulness and Usability

The usefulness of the system is measured by questions 1 to 6, 13 and 14 in the questionnaire. The average of usefulness ratings was 3.51 (3.s.f.) with a standard deviation of 0.919. One standard deviation of the data from the mean (containing 68% of the data points) put the usability of the system in the range 2.59 through 4.43. The percentage of positive usefulness ratings (those greater than 3 in the questionnaire) was 55.9%, and the percentage of negative ratings (those below 3 in the questionnaire) was 22.6%, as calculated from each individual question in the questionnaire.

The usability of the system is measured by questions 7 through 12 in the questionnaire. The average of usability ratings was 3.46 (3.s.f.) with a standard deviation of 0.810. One standard deviation of the data from the mean put the usability of the system in the range 2.65 through 4.27. The percentage of positive usability ratings was 50.0%, and the percentage of negative ratings was 23.8%.

A box plot was produced showing the mean usefulness ratings and usability ratings per user and is shown in Figure 7.1. This shows that the spread of usability ratings was smaller than that of usefulness, but that the means and interquartile ranges were similar.

In section 4.5 the success criteria for both usefulness and usability is defined as 'a significant positive average rating' for the relevant questions. Both the mean ratings and standard deviations for both the usefulness and usability questions were very similar - both show a slightly-above-neutral mean rating of the system, and the standard deviation shows the data is moderately distributed over the range of possible ratings (it was not highly concentrated or well distributed).

Because of the spread of the results it is not all-together clear whether there is a significant positive view of the usefulness and the usability of the software. However, as the mean is above the neural point of 3 by 0.51 and 0.46 respectively (representing 1/8th of the range of possible ratings) it is justified to state that the software was rated as having neutral or greater usefulness and usability. This does not fulfil the initial success criteria from section 4.5, but still shows the software developed was not rated 'useless' and so has potential to become useful.

7.2.2 Machine Learning

A mean value of usefulness as rated by each participant was calculated using the values from questions 1 through 6, 13 and 14 and plotted. Linear regression was then carried out to identify whether a positive correlation existed showing

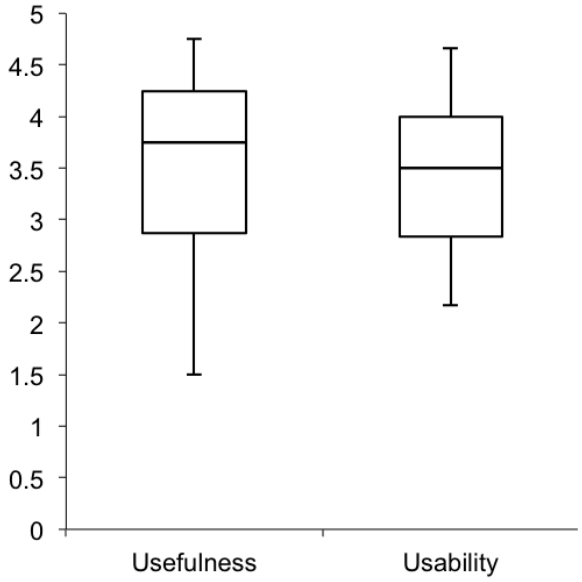

Figure 7.1: Box plot showing usefulness and usability ratings derived from calculating a mean usefulness and usability rating from each user.

that participants were indicating they would find the application more useful as more participants used it before them. The Pearson product-moment correlation coefficient (PCC) was calculated for this dataset to establish the strength of the correlation using the following equation:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

In this equation, `n` is the number of participants, `x` is the participant ID, and `y` is the mean usefulness rating calculated for an individual participant. The PCC gives a value in the range of `-1`, which indicates a perfect negative correlation, to `+1`, which indicates a perfect positive correlation, with zero indicating no correlation at-all. Using this equation the correlation coefficient for this dataset was found to be 0.506.

A graph showing usefulness plotted against the participant number is shown in figure 7.2 Using linear regression a line of best fit was added to the graph with the equation $y = 0.075x + 2.68$.

These results show that the application was successfully learning. As discussed

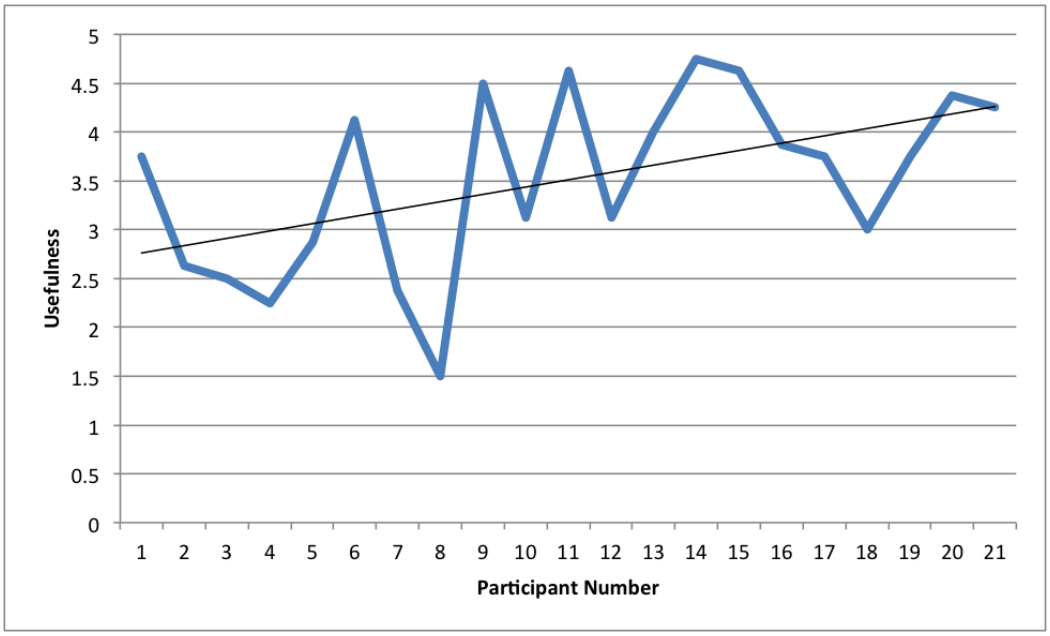

Figure 7.2: Graph showing Usefulness as rated by participants against participant ID with a trend line.

in section 4.5, the effectiveness of the machine learning was evaluated according to a positive trend in feedback from participants as the number of participants who have used the application prior to them increases. A positive trend line can clearly be seen in the graph showing rated usefulness against the number of preceding participants. The correlation coefficient indicates that there is a moderate positive trend, which implies that the application was learning and its usefulness improving as more participants used it. This is therefore a success in the context of this project.

The graph does show that the data collected is 'noisy' with a lot of variation in the ratings given by different participants. It is speculated that this was caused by the different levels of expectations that participants had. The questionnaire was worded to remind participants that they should rate the system in the context of them having no other access to the internet, but it is likely some participants considered this more than others whilst rating the application, sometimes making comparisons to other services from well-known consumer products that they have used.

The correlation coefficient was calculated individually for each of the questions that measure usefulness to establish if any of the specific questions yielded a particularly low or high correlation coefficient when measured against the number of preceding participants. The results are presented in table 7.1, which shows that

all of the questions showed a positive correlation, with varying degrees of strength. The results from questions four and thirteen are of interest: question four has a much lower correlation coefficient than other questions, and question thirteen has a relatively high correlation coefficient.

Question four looked at whether the application would improve the effectiveness of a user at finding information without access to the internet, and this result would indicate that correlation between this and the number of participants who have used the application previously is very weak. This means that as more users used the application, the application did not appear to improve the way users felt about their effectiveness at finding information without access to the internet.

Question thirteen investigated whether a user felt the information returned was useful, relevant and answered the question asked. A Correlation Coefficient value of 0.617 shows a moderately strong correlation, which shows that the information returned did improve in terms of its usefulness and relevance to the question asked.

Question ID	PCC
1	0.424
2	0.540
3	0.294
4	0.125
5	0.371
6	0.425
13	0.617
14	0.240

Table 7.1: Table showing the Pearson product-moment correlation coefficient of the answers given for the individual usefulness questions against the number of preceding participants.

7.2.3 Additional comments from participants

No space was made on the questionnaire for participants to write additional comments or thoughts on the system, however two participants did make a comment at the end of their forms, both of which addressed the same issue. These comments were:

'I thought the detail generated from the answers to the describe questions were really detailed and informative, although displaying the units in other answers would have been helpful.' - Participant fourteen.

‘Answers with units would be helpful to understand values given as answers’ - Participant nine.

Although this issue was noted before the experimental evaluation, the significance of not having units had not been fully considered and it was helpful to have this indicated by the participants. These can’t be included in any formal analysis due to not asking for a comment from all participants.

7.3 Data loss

It was initially planned to keep log files during the experiment containing the questions asked, answers provided and the ratings from a user. This would allow for the possibility of further analysis such as the identification of specific questions that caused users a problem or questions where the answer returned changed as more participants took part. However due to a bug where log files were over-written instead of having new entries appended to them, log files for participants up until participant number 14 were not stored. The seven remaining log files were too small a dataset to analyse.

8 Evaluation

This Chapter aims to provide an evaluation on the experimental evaluation that was run and the software that was developed. The experiment will be evaluated based on the results, and the software will be compared to the original requirements in section 4.2.

8.1 Evaluation of Software Developed

The success of the software developed according to the experimental evaluation has been discussed in the Results and Discussion chapter in section 7. However it's also important to check the software against the requirements originally proposed. The software has been compared to the User Requirements from section 4.2.2 in table 8.1. In this table it is shown that all user requirements for the system were met apart from user requirement 2 which is partially met. User requirement 2 refers to the fact that units are not always returned with answers and this is a usefulness failure in the system. A proposal for future work to fix this is included in section 9.1. The application meets all other user requirements fully.

The software is also compared to the functional requirements given in section 4.2.3 in table 8.2. This table shows that the application developed fulfils all of the system requirements.

Finally, the software is compared against the non-functional requirements proposed in section 4.2.4 in table 8.3. In this table it is shown that the application meets all non-functional requirements except NF.2, which is that the application returns the correct answer for 90% of queries. This requirement was marked as not met, as the data that would have been used to evaluate it was lost, as described in section 7.3. It was suggested that the ratings provided for application usefulness in the questionnaire would have been higher if this requirement was met.

Non-functional requirement NF.2 cam be improved by extending the amount of data that the application can retrieve from the data sources. As described in section 4.3.2 some information for a Geographical entity in the DBPedia Ontology is not queryable in the current implementation, which simply pulls a list of top-level

parameters about an entity. This could be improved by adding support for complex Ontological queries using the SPARQL interface [**?**], and is discussed in the Future Work chapter in section 9.6. Additional data sources could also be added to the application to increase the amount of information available for the application to utilise when answering questions. This is also discussed in the Future Work chapter in section 9.5.

8.2 Evaluation of Experimental Evaluation

The experimental evaluation suffered from issues caused by the pool of participants used. As discussed the data was extremely noisy and it was speculated that this was caused by varying levels of expectation from participants. This could have been fixed by spending more time with each participant explaining the expectations they should have of the software and a typical situation in which it would be used.

The participant pool contained 21 individuals and was skewed towards students in the age range of 18-24 who used the internet multiple times per day. This is not representative of the range of possible users of the application developed, notably the personas developed in appendix A which were used to drive the user requirements. As the participant pool was skewed towards a young student population with regular access to the internet it is suggested that some had difficulty imagining themselves working in the context of not having access to the Internet.

Each of the 21 participants in the experimental evaluation were asked to identify 16 pieces of information. The application learns more as each participant takes part and it would have been interesting to see if the trend of increasing ratings of usefulness continued with a larger number of participants. Time restrictions limited the number of data points that could be collected from each participant: with 16 pieces of information to identify the experiment took around 10 minutes during a test run. It would have been preferable to have more data points from asking participants to identify more information as this would perhaps have reduced the noise in the data due to a single poor result having less of a part of the whole experiment. A suitable expansion of the task would be to ask participants to identify 50 pieces of information instead of 16 which would extend the experiment to take approximately 30 minutes. Unfortunately this would make it more difficult to recruit participants without a remuneration for their time.

The experiment showed a usefulness trend that indicated the application was learning and improving, however as was previously mentioned the data was noisy. Ideally additional participants would be used to verify the quality of answers

provided by the application continues to improve. It is suggested that a suitable expansion would be to increase the number of participants to between 30 and 40. This would mean the experiment had more participants who used the application in the later stages of its learning process.

8.3 Conclusion of the Evaluation

The software developed has been evaluated according to the original user requirements and non-functional requirements of the system in section 8.1. It was shown that the system satisfied all of the requirements except two, one of which it partially satisfied. From this it is concluded that the development of the software was majoritively successful, but with space for future improvement especially in the area of answer quality. Suggestions for improvements to the system including those that will improve the quality of answers are provided in the Future Work chapter in section 9.

The experimental evaluation was evaluated with a focus on the participants used and the data collected. Some issues were found, such as that the participants had different expectations of the quality of answers they could expect and different perceptions of the situations that the application would be used in. The experiment was limited by time and cost: ideally more participants would be used and more data points would be collected. However, valuable data was collected showing that the application partially met its usefulness and usability goals and a correlation was found showing that the application was learning, making the experimental evaluation a moderate success.

ID	Discussion	Conclusion
1	Two types of questions were accepted: questions where a property was requested of a geographical entity; and questions where a user could ask the application to describe an entity.	This requirement was met.
2	Users were able to ask questions about a specific property, fact or statistic, however in some cases the units were not returned which reduced the usefulness of the answer. This occurred in a significant number of cases, such as whenever a measurement of an entity was requested, so the requirement wasn't fully met.	This requirement was partially met.
3	The Twilio service and API was used to create an SMS interface to the system.	This requirement was met.
4	Users were able to make queries quickly by entering them in natural language form, and a response was returned within a a few seconds.	This requirement was met.
5	A ratings system was implemented to allow a user to give feedback on the last answer they received, which was subsequently used to improve the quality of future answers.	This requirement was met.
6	Help and tutorial functionality was provided in the system and was available for users by simply sending a message with the world `help`. This was not tested during the experimental evaluation however, as a tutorial was given in the instructions for the experiment. Feedback from the experiment did show that users quickly learnt to use the system though, indicating that they were able to learn the system themselves.	This requirement was met.
7	A privacy protection system was implemented. Participants in the experiment were not made aware of this as the experiment focussed on answer quality, however they were comfortable with the privacy they had for their experimental data (explained on a consent form).	This requirement was met.
8	Natural Language Processing was added to the application to allow users to make queries in natural English sentences. Users used this in the experimental evaluation and it was rated neutrally or above as part of the usability questions.	This requirement was met.

Table 8.1: Evaluation of User Requirements

ID	Discussion	Conclusion
1	The system has a Natural Language Processing (NLP) module which processed queries in natural language.	This requirement was met.
2	The system has an SMS interface allowing it receive queries via SMS and to then send replies via SMS.	This requirement was met.
3	The system uses a number of data sources including DBPedia, which is able to return specific facts about an entity as well as a general description.	This requirement was met.
4	The system can parse queries using the NLP module to identify the entity that is been queried about and the parameter a user is looking for.	This requirement was met.
5	The system uses the general descriptions of entities in DBPedia to support providing general descriptive information on an entity.	This requirement was met.
6	The system uses semantic comparison libraries to match words that are semantically related.	This requirement was met.
7	The system learns from feedback sent to it about whether a particular answer is of high or low quality and then applies this to future queries.	This requirement was met.
8	The system has built in documentation that can be retrieved by sending the message 'help'	This requirement was met.
9	The previous question asked is stored in the User object in the database to support queries that rely on user state (such as rating a previous answer).	This requirement was met.
10	The system was not overly complex to implement as appropriate tools and languages were used. A modular structure was used to make the application easily extendable for future work.	This requirement was met.
11	The system obscures a user's phone number via hashing.	This requirement was met.

Table 8.2: Evaluation of Functional System Requirements

ID	Discussion	Conclusion
NF.1	The time from receiving a query to sending a response was typically under 10 seconds during development, which is less than the 20 second fit criterion.	This requirement was met.
NF.2	Due to the loss of log file data it is not possible to analyse the question logs to check against this criteria. However it is speculated that usefulness ratings from the experimental evaluation would have been higher than they were if 90% of questions were answered correctly first time so it is assumed this requirement was not met. The bug that led to data loss was resolved after the evaluation with participant 14 took place.	This requirement was not met.
NF.3	The ratings system implemented did prevent poor answers been returned after they were rated as such. Examples were given in section 6.6.2.	This requirement was met.
NF.4	The help information was not used during the experimental evaluation however users did not need further assistance when using the application beyond the instructions provided for the experiment, and the instructions provided were based on the instructions in the application.	This requirement was met.
NF.5	Participants used natural language to ask questions of the application during the experiment.	This requirement was met.

Table 8.3: Evaluation of Non-Functional System Requirements

9 Future Work

In this chapter solutions for some of the issues addressed in the evaluation will be presented as work that can be undertaken in the future to improve the software developed.

9.1 Obtaining and presenting units for measurements

One of the major issues with the application developed is that units are not returned with all measurements. This is a direct result of the quality and inconsistent format of the data in the data sources. For some geographical entities units are stored in the field when the field type is a string, however in other entities the unit is stored in the key, for example `height_m` is used to represent the height in metres. It should be possible to customise the processor for each data source in the source-specific module to handle this in the future.

9.2 Improving the Natural Language Processing module

It was noticed both during development and informally during the experiments that the Natural Language Processing module isn't yet at the standard that a user expects. Although queries can be made in natural language, the structure of the question must be extremely simple. This is because of the way the property a user is requesting is identified, where additional unnecessary words in the query cause the process to fail. There were two example of this, one that arose during testing and one from the log files collected from the experimental evaluation. The first example was where the word `please` was added to a query, for example `How tall is the Eiffel Tower please?` which resulted in the application searching for the property `tall please`. The second example from the experimental evaluation

occurred when a user was asked to identify the `currency that we use in England`. A typical query was `What currency is used in England?`, which resulted in the application searching for the property `currency used` instead of just `currency`. This also resulted in the application failing.

In future work the natural language processing module can be improved to cope better with noise, perhaps through the use of a more complex third-party library.

9.3 Expanding the range of subjects from just Geography

Geography was selected as the topic area to perform research on in this project because it was a topic with few ethical issues, as discussed in the introduction in section 2.3. If the ethical issues could be addressed and resolved, interesting future work includes expanding this application to work with Wikipedia resources from a larger number of subject areas. There would also be some engineering issues: the current software is dependent on knowing that a query is about a geographical entity, as the Natural Language Processing module, described in section 6.4, only looks for an entity that can be classed as a Place. Changing this is to a different type of entity is trivial: the Place class must be replaced with the DBPedia class that describes the new type of entity that the application will work with. However, as the range of possible entity types expand there will be issues in identifying the specific entity that a user is inquiring about if a query contains multiple entities or concepts.

9.4 Add support for multiple languages

This has been discussed in section 6.9 in the implementation as detail was given as to where it would fit into the application.

9.5 Adding additional data sources

Currently only Wikipedia and DBPedia are used as data sources for the application. Additional databases for specific topics could added to the software to improve the quality of answers for specific areas. Adding these data sources would be simple: a module would be created that follows the interface specification given in 6.5. An

example of a data source that could be added is the Peakware World Mountain Encyclopedia [**?**] through a possible collaboration with the owners. This would improve the information available in the application specifically about mountains. An ideal data source uses simple and descriptive names for keys in the database to allow the semantic analysis to link the terms from a users query to the key names.

9.6 Using SPARQL to perform complex queries in DBPedia

One area where significant progress can be made in the software is with queries to the DBPedia ontology. In section 4.3.2 the available interfaces to DBPedia are discussed and the issues with creating SPARQL queries presented. The main issue is one of software complexity: taking a query in natural language form and then constructing a formal Ontology query using the SPARQL interface [**?**] is difficult as terms in the query have to be matched to classes in the Ontology. However if this were to be achieved the results that the Ontology could return would be significant of significantly higher quality as data that is abstracted into linked objects could be retrieved. An example of this is the query `Who is the mayor of London?`. The object `London` in the DBPedia Ontology does not have a field `Mayor`, but instead a pointer to another object called `Mayor_of_London` which contains the data requested. This was beyond the scope of the original project due to the time constraints, as this will not be a quick feature to implement.

10 Conclusion

This project presented the design and development of an application for providing automated answers to questions relating to geography via SMS, targeted at an audience who do not have access to information via other means such as books or the Internet. The application uses machine learning to improve the quality of the answers it provides over time. The development of the application included researching and implementing Natural Language Processing, Machine Learning, and a system for semantically matching related phrases and keywords.

The application was evaluated through an experimental evaluation where participants were asked to use the application and then complete a questionnaire. The questionnaire specifically asked questions that measured how useful the application was, and its usability. The application was also compared to the user requirements, functional system requirements and non-functional system requirements.

The overall implementation of the system is viewed as being a success, with the application fulfilling all of its functional system requirements and the majority of its user requirements and non-functional system requirements. Participants in the experimental evaluation rated the application positively with regard to usefulness and usability, and it was shown that the machine learning in the application was effective at improving the quality of answers

One of the more complex parts of this project was extracting high quality information from the data sources used. This affected the answer location system, sometimes resulting in the application being unable to find a simple answer, or in the case of a question about a measurement of something (such as height) being unable to return the units with the number. Another issue that was faced came from the APIs that were used in the project, some of which changed a few times. This was fixed by developing the application in a modular way such that modules could be adjusted and then swapped in and out.

There was an awareness of ethical issues throughout this project, specifically focussing on protecting the privacy of users. This was well researched in the early stages of the project.

A key point that was learnt during this project is that locating information in

third-party data sources from natural language descriptions of the data required is a non-trivial problem, however that machine learning can be used to match natural language to data in data sources. Another learning point that stems from this was that, even with good machine learning, the quality of responses is limited by the quality of the data sources used.

There is substantial room for future work to both expand this project and fix some of the limitations, including improvements to the Natural Language Processing and Machine Learning Systems. However the application developed did meet its goals and despite its limitations, a tool was produced that is capable of providing helpful information to a user.

APPENDICES

A Personas

A.1 Nanjala

A.1.1 Demographics and Profile

- Female, aged 15
- Living in the village of Amuria, Uganda, 300km from the capital Kampala. Internet access is difficult and expensive to obtain in Amuria, though there is good mobile network coverage.
- Currently in general education at her village school.
- Nanjala is curious and wants to be able to ask questions and to retrieve and read more information at her own leisure. She prefers to be told about something than to learn facts.

A.1.2 Average Day

In a typical day, Nanjala will rise early to help her family with housework chores, before she prepares for school. Nanjala wants to become a teacher. At school, she studies Science, Maths and Geography, and works hard for her classes and on her homework. Nanjala's school has a small library and a large body of students, meaning that books are often unavailable when Nanjala would like to look something up. This increases the amount of time that her homework takes, and can even result in her not having access to information when she goes home.

A.2 Cedric

A.2.1 Demographics and Profile

- Male, aged 27
- Lives in Zurich, Switzerland, and goes on regular hiking trips, from 3-4 day trips around Europe to longer trips to the South American Andes and southern Asia.
- Works in a service role in the city and so has little disposable income to spend on guide books for the many places he visits.
- Cedric is a competent albeit not professional hiker, often going off-track and summiting 'difficult' peaks or reaching famous camping grounds. He likes to be spontaneous, only deciding which peaks he will attempt when he arrives.
- Cedric is very conscientious and wants to contribute his own knowledge to the world, especially to projects that help him.
- Cedric values his privacy is concerned about how much information he gives out online.

A.2.2 Average European Trip

A typical trip will involve an intercity train from the Zurich Hauptbahnhof (the main station) to the city closest to the range Cedric will be hiking. He takes the train because it tends to be cheaper than flights. Once in the area, Cedric will take local trains and buses to get into the mountain ranges. There, he will use the local hiking trail network signs and conversations with the locals to select a peak or route. Once Cedric has got moving and is reaching higher ground, he uses a combination of his view and the signage to identify other peaks that he would like to wander around.

Cedric takes a non-smart phone with him because a) the battery life on his smart phone is not long enough for his whole trip, whereas his non-smart phone lasts a week at a time, and b) his smart phone would not be useful anyway, since the internet is so intermittent. He still wishes he could get information such as whether there are maintained routes on the slopes of a given peak or the height of a peak that he sees without having to phone friends back in the city.

B Sample Consent Form

System to access knowledge using Machine Learning and SMS - Usability and Usefulness Study

Informed Consent - Introduction

This study aims to investigate the usability and usefulness of a system developed to provide access to knowledge using machine learning and SMS. The system aims to provide high quality answers to questions asked of it, where the questions relate to either a description or property of a geographical entity (for example a city, river or mountain). The system aims to learn where mistakes have been made and improve by receiving feedback from the user on the previous answer.

Form: please complete this section before the study.

I, _____, agree to participate in this study. I have been briefed on and understand the purpose and goals of the project.

I understand that information will be collected from me during this study in the form of my answers to a questionnaire. Additionally, I understand that my interactions with the application will be recorded, specifically the questions I ask, answers returned and my quality ratings of answer. I understand that this information will be treated confidentially and only Sam Heather and the project supervisor (Lilian Blot) will have access to the original data in its original format. When the information is processed, shared, described or interpreted I will not be personally identified.

I understand that I can withdraw from the study at any point without giving a reason.

Date: _____Signature: _____

Researcher's contact details: Sam Heather, sam@heather.sh

Supervisor's contact details: Lilian Blot, lilian.blot@york.ac.uk

Post experiment consent form - please complete this section after the study.

□ I have been suitably debriefed about the system and my performance.

□ I understand I can withdraw my data from the study at any time.

□ I have not been forced to take part in this study.

□ The Researcher has treated me with respect and answered all of my questions.

For the researcher: Participant ID: _____

C Experimental Evaluation Instruction Sheet

System to access knowledge using Machine Learning and SMS - Usability and Usefulness Study – Instructions

Introduction

This study aims to investigate the usability and usefulness of a system developed to provide access to knowledge using machine learning and SMS. The system aims to provide high quality answers to questions asked of it, where the questions relate to either a description or property of a geographical entity (for example, a city, river or mountain). The system aims to learn where mistakes have been made and improve by receiving feedback from the user on the previous answer.

Your Task

With consideration to the goals of this study in mind, you should attempt to locate specified information using this tool. You should ask questions in natural language form as you would a friend, for example:

> "How tall is the Eiffel Tower?".
> "What's the population of Mexico city?"

You will either be asked to locate a specific property of information (such as the height of a monument), or to locate general information. You should note that if the primary method of finding information (that of asking a question) should fail, you can always ask for a general description of the entity using the syntax:

> describe X

The system should learn which answers are of a particularly high or low quality. To allow it to do so **please rate every answer you receive** with a star rating between 1 and 5 by replying with a message such as:

rate 1	(for a poor quality answer)
rate 3	(for a neutral quality answer)
rate 5	(for a high quality answer)

After the experiment you will be asked to fill in a survey about your experience using this application.

Questions to Ask

Please locate the following information using the application. You are encouraged to construct your questions using a variety of terms, not just those provided below. You are not required to write down or record the information you find. If no answer is returned, you may ask again in a different way, however if an answer is returned you should not ask again, even if the answer is obviously incorrect. Please remember to rate every answer you receive.

Please identify:

1. the height of the Eiffel Tower
2. the height of another monument of your choice
3. the number of people living in London.
4. the depth of the English channel
5. the year that the Eiffel tower was built
6. general information on Paris
7. the elevation of mount Everest
8. the currency that we use in England
9. The height of the Leaning Tower Of Pisa
10. the length of the amazon river
11. a general description of Snowdon mountain.
12. the population of a country of your choice?
13. the GDP of Russia
14. the country in which Barcelona is a city
15. the area of a country of your choice
16. the discharge of a river of your choice

D Sample Questionnaire

Please fill in the following questionnaire with regard to your experience of the system that has been developed.

#	Question	Agreement 1 = strongly disagree 3 = neutral 5 = strongly agree
1	Using the system would enable me to locate information more quickly when without access to the internet.	1 2 3 4 5
2	Using the system would improve my performance when working or researching without access to the internet.	1 2 3 4 5
3	Using the system would increase my productivity when performing casual research without access to the internet.	1 2 3 4 5
4	Using the system would enhance my effectiveness at finding information without the internet.	1 2 3 4 5
5	Using the system would make it easier for me to find information without access to the internet.	1 2 3 4 5
6	I would find the system useful in day-to-day life when without access to the internet.	1 2 3 4 5
7	Learning to operate the system was easy for me.	1 2 3 4 5
8	I would find it easy to get the system to do what I want it to do.	1 2 3 4 5
9	My interaction with the system would be clear and understandable.	1 2 3 4 5
10	I found the system to be flexible to interact with.	1 2 3 4 5
11	It would be easy for me to become skillful at using the system.	1 2 3 4 5
12	I found the system easy to use.	1 2 3 4 5
13	The information returned was useful, relevant, and answered the question asked.	1 2 3 4 5
14	I trust the answers returned by this application.	1 2 3 4 5

Please turn over for demographic questions.

15. Your Age:

☐ 18-24 years old
☐ 25-34 years old
☐ 35-44 years old
☐ 45-54 years old
☐ 55-64 years old
☐ 65-74 years old
☐ 75 years or older

16. What is your occupation?

☐ Employed for wages
☐ Self-employed
☐ Out of work
☐ A homemaker
☐ A student
☐ Military
☐ Retired
☐ Unable to work
☐ Other: ……

17. Frequency of Internet Usage:

☐ Usually more than once per hour
☐ Multiple times per day
☐ Once per day
☐ Multiple times per week
☐ Once per week
☐ Less than once per week

For the researcher: Participant ID: ____