University of Essex – Computer Science

Interim Report

Open Domain Question & Answer System

Osama Rahman – 1304349

John Gan

Diego Perez Liebana

Contents

1.0 Introduction

This report will outline current and future progress of the project. Currently implemented and future parts of the system will be discussed in high technical detail, along with technical diagrams to show how parts of the system interact with each other. The open source projects used will also be discussed in detail. Development techniques will also be discussed, as well as an in-depth literature review.

The overall aim of the project is to create an Open Domain Question and Answer system, a system which will be able to answer simple questions such as “Who is Elon Musk?”, or “What is the capital city of Peru?” While no system can be truly open, we are able to develop systems which can answer vast amounts of factual questions with great accuracy. These systems include natural language engines, an information retrieval aspect, and will be connected to either a database or knowledgebase of some kind.

2.0 Project Management

2.1 Project Goals – Revisited

Core Goals:

- Create a user-friendly GUI to input and output text and data
- Parse natural language user input into a readable query
- Use the query to retrieve data from a database/knowledgebase
- Parse the information from the database/knowledgebase into a natural language reply

Secondary Goals (To be completed if possible / have time):

- Implement a voice API to allow a user to ask a question through speech
- Implement a large data dump, to give the system a more open domain
- Implement a way to answer basic maths questions
- Implement a weather API for the system to report back weather
- Implement a world clock API for the system to report global times

Personal Goals:

- Enhance my programming knowledge and skills
- Further educate myself in the field of artificial intelligence
- Enhance my project management skills and knowledge
- Enhance my professional skills

[1] Goals from the Initial Report

The project goals give a general overview of what is expected to be accomplished during the project period. Above is a rundown of the project goals, which were originally outlined in the initial report. The goals have largely stayed the same. As development progresses the secondary goals will become more feasible.

One of the secondary goals was to use a large data dump for the information retrieval. This proved to be impractical as it required too much disk space, and would have likely slowed down the system. Instead of "*Implement a large data dump to give the system a more open domain*" [1], the goal has been pivoted to "*Connect to the DBpedia knowledgebase to broaden the domain of the system*". While this greatly reduced the size of the system, and will likely increase speed across a range of devices, the system will require internet access to retrieve the information. Considering the uses of this

application and its types of users, this is an acceptable trade-off. Other goals could possibly change throughout the project, and more may even be added. The personal goals will remain the same throughout.

2.2 Updated Goals

Core Goals:

- Create a user-friendly GUI for the user to interact with the system.
- Parse natural language user input into a SPARQL query
- Use the query to retrieve data from a DBpedia
- Parse the retrieved information into a natural language reply.

Secondary Goals (If feasible):

- Implement a voice API to allow a user to ask a question through speech
- Connect to the DBpedia knowledgebase to broaden the domain of the system
- Implement a way to answer basic maths questions
- Implement a weather API for the system to report back weather
- Implement a world clock API for the system to report global times

Figure 1- Updated Objectives

Updating the objectives list allows for a better understanding of the system, and what is needed for the project to be successful.

2.3 Current Progress

In its current state the project has started unexpectedly. Initially the starting point was going to be developing the natural language parser, however after some more research and testing the conclusion was made that a connection to DBpedia needed to be made first. This is because the SPARQL query language needed to be familiarised before the natural language to SPARQL parser could be implemented.

With access to DBpedia the system can retrieve some information from the knowledgebase, using SPARQL. However, the system currently requires the tester to input a prefix, and a property string to access specific information. The system starts out by asking “What would you like to know about”, the tester then enters something like “University of Essex” which will be appended to a URL, the prefix and property are entered and a result is returned. The below snippet provides more clarity.

```
Hi, I'm TonAI. What would you like to know about?
University of Essex

Enter a prefix (rdfs, dbo)
rdfs

Enter a property input (comment, birthYear)
comment
The University of Essex is a British public research university in Colchester, Ess
ex. It was established in 1963 and received its Royal Charter in 1965. The univers
ity's largest campus is located within Wivenhoe Park in the English county of Esse
x, less than a mile (1.6 km) from the town of Wivenhoe & 2 miles (3.2 km) from the
 town of Colchester. Apart from the Wivenhoe Park campus, there is a rapidly devel
oping campus in Southend-on-Sea (Essex's largest town), and the East 15 Acting Sch
ool is based in Loughton. The University's motto, Thought the harder, heart the ke
ener, is adapted from the Anglo-Saxon poem The Battle of Maldon. The university en
joys collaborative partnerships with a number of institutions across the eastern r
egion. These are University Campus Suffolk, Colchester Institu
```

Figure 2 - Current state of the system

With the natural language parser not yet implemented, the system is still very much in its infancy. In the final version, it will not be expected for a user to input the prefix and property. They will be able to simply ask a question and receive an answer. This stage needed to be developed for the design documentation to be precisely completed.

2.4 Waterfall Methodology

As mentioned in the initial report, the waterfall methodology is being used for this project. It is the most suitable methodology for a project of this nature as it necessitates a strict set of requirements as well as a clear design solution.

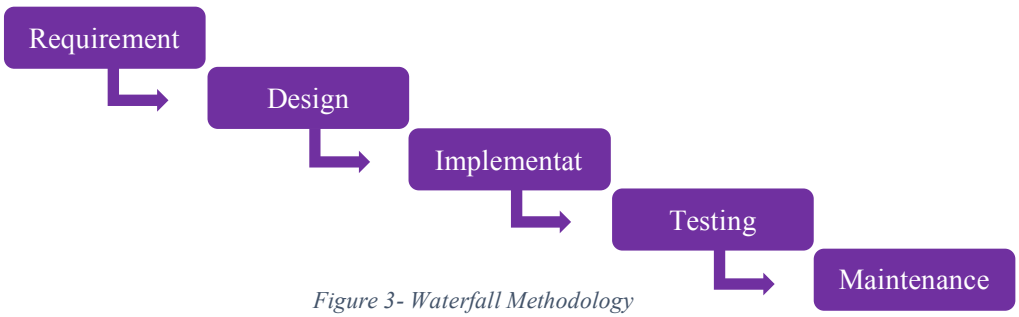


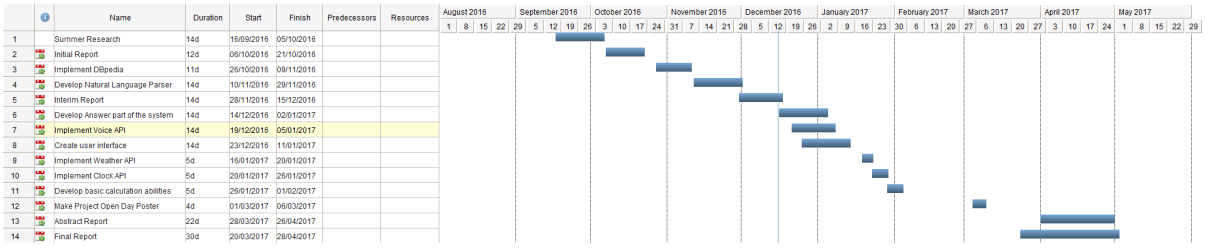
Figure 3- Waterfall Methodology

The above diagram shows the main stages within the waterfall methodology. With the requirements and design completed, as in this document, it is now time to start the implementation stage. The requirement document will outline everything that is needed to be completed within the project. In regards to the design, some actual programming was needed to be able to specify the design principles which are to be followed. Once implementation is completed, testing will be conducted on the program. Small amounts of testing and debugging will need to completed during the implementation stage to make sure specific parts of the system work. However, the during the testing phase a higher

level of testing will be required. Testing which will include having actual users use the system. After the testing phase is complete, and the system is working to the requirements and design specification, it will need to maintained. Maintenance will include upgrading aspects of the system to newer standards, and making sure it continues work on newer machines.

2.5 Gantt Chart

Before development started a Gantt chart was created, setting out clear goals and deadlines. However, the feature which was to be developed first had to be pushed back due it relying on another feature. The updated Gantt chart, featured below, gives a view of the project plan. With other deadlines to meet this project is a little behind. There were two main aspects which are yet to be worked on. Those being the natural language parser, and the answering part of the system. While the system can retrieve some information, it is unable to parse or filter this information. The initial Gantt chart shows that the natural language parser was to be built first. It was appropriate to postpone this as SPARQL needed to be familiarised before any natural language to SPARQL parsing could be made. A connection to DBpedia also needed to be made to test SPARQL, and to retrieve information using a parsed query.



	Name	Duration	Start	Finish	Predecessors	Resources
1	Summer Research	14d	16/09/2016	05/10/2016		
2	Initial Report	12d	06/10/2016	21/10/2016		
3	Implement DBpedia	11d	26/10/2016	09/11/2016		
4	Develop Natural Language Parser	14d	10/11/2016	29/11/2016		
5	Interim Report	14d	28/11/2016	15/12/2016		
6	Develop Answer part of the system	14d	14/12/2016	02/01/2017		
7	Implement Voice API	14d	19/12/2016	05/01/2017		
8	Create user interface	14d	23/12/2016	11/01/2017		
9	Implement Weather API	5d	16/01/2017	20/01/2017		
10	Implement Clock API	5d	20/01/2017	26/01/2017		
11	Develop basic calculation abilities	5d	26/01/2017	01/02/2017		
12	Make Project Open Day Poster	4d	01/03/2017	06/03/2017		
13	Abstract Report	22d	28/03/2017	26/04/2017		
14	Final Report	30d	20/03/2017	28/04/2017		

Figure 4 - Updated Gantt Chart

3.0 System Requirement Specification

3.0.1 Definitions and Abbreviations

Term	Definition
The System	The Open Domain Question and Answer System being built.
QA System	Question and Answer system.
Natural language	Language in which people communicate.
Information Retrieval	The capability to computationally retrieve information from a source.
Knowledgebase	A facility to store information which can be extracted and acted upon.
Resource Description Framework (RDF)	A data format for linked data and the semantic web
Natural Language Tool Kit (NLTK)	The natural language library which is to be used for the natural language processing.
DBpedia	The knowledge base being used to retrieve information.
SPARQL	The RDF query language used to Query DBpedia

3.1 Purpose

This system requirement specification (SRS) will provide a detailed description of the Open Domain Question and Answer System being built. The aim of this document is to give the reader an understanding on how specific parts of the system will work, the systems constraints, and possible future features.

3.2 Project Scope

The system is intended for users who wish to gain some form of information using natural language. This type of system is expected to accompany users throughout the day on their desktop operating system. For example, a history student could be writing a report and ask “What year did World War 2 start?”, or a reporter could ask “Where is Elon Musk from?”. With such a system, a specific application scope is difficult to define as there are many, although general, uses for it. Essentially the system is for anyone who requires to gain simple information quickly and effortlessly while using a more human method of communication, natural language.

3.3 Design

This section will discuss the individual aspects of the system, and how they work. Included will be the modules being used, and core parts the system needs to function correctly. The design of the document is a crucial factor when working using a waterfall methodology. Each aspect of the system must be carefully considered and executed with precision.

3.3.1 Natural Language Parser

The purpose of this part of the system is to parse the natural language question into a machine readable SPARQL query. This will be accomplished using the Python NLTK library. The NLTK library allows the natural language question to be generated into a tree structure, which will be used to create POS (parts of speech) tags. These tags will then be analysed to create a query. Tags include nouns, verbs, adjectives, etc.

More research than development has been made on this aspect of the system. Per the Gantt chart, this was supposed to the first part of the system to be developed, however it was more appropriate to first create a connection to DBpedia.

The parser will also be used to return information retrieved from DBpedia in the form of a natural language reply. For example, if asked “What is the capital city of England”, the system would reply “The capital city of England is London”, as opposed to simply “London” which would be the initially retrieved information.

3.3.2 Information Retrieval

The information retrieval aspect of the system relies on the natural language parser to form an accurate SPARQL query. Take the following SPARQL query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?dbo
WHERE {<http://dbpedia.org/resource/Elon_Musk>} dbo:birthYear ?birthYear
```

Figure 5 - SPARQL Query to retrieve the birth year of Elon Musk

The above query will reference DBpedia, and return the birth year of Elon Musk. To generate such a query a user would ask the system a question such as “When was Elon Musk born?”, or “What year was Elon Musk born?”

SPARQL uses RDF prefixes [2] to separate certain data sets. Here the *dbo* prefix is used to return the value of *birthYear*. If we were to replace *dbo* with *rdfs*, and *birthYear* with *comment*, DBpedia would return a paragraph comment of Elon Musk. Following the URL will take you to the Elon Musk DBpedia entry, which contains prefixes and other datasets used for information retrieval.

DBpedia relies on Wikipedia. One shortfall of this system is that for it to work seamlessly the Wikipedia articles would have to be constantly kept up to date. The accuracy of the Wikipedia article is also of concern. However, with its coming of age, and the fact Google, Apple, and many other reputable companies have systems which reference it, it can be concluded that most articles are indeed quite accurate.

3.3.3 GUI

For the system to be intuitive and for interaction to be seamless the GUI will resemble a chat like interface. It will look like an interface you would find in messaging apps used to communicate with other people. This style also allows for users to look for answers of past questions. A familiar interface allows for users to quickly grasp how to use the system, and is likely to aid in user retention. The design of the GUI will be minimal and up to date with modern software design trends. The below image shows a mock-up of how the GUI will look. It should be noted that some aspects of the GUI are likely to change during development, and are dependent on python GUI modules.

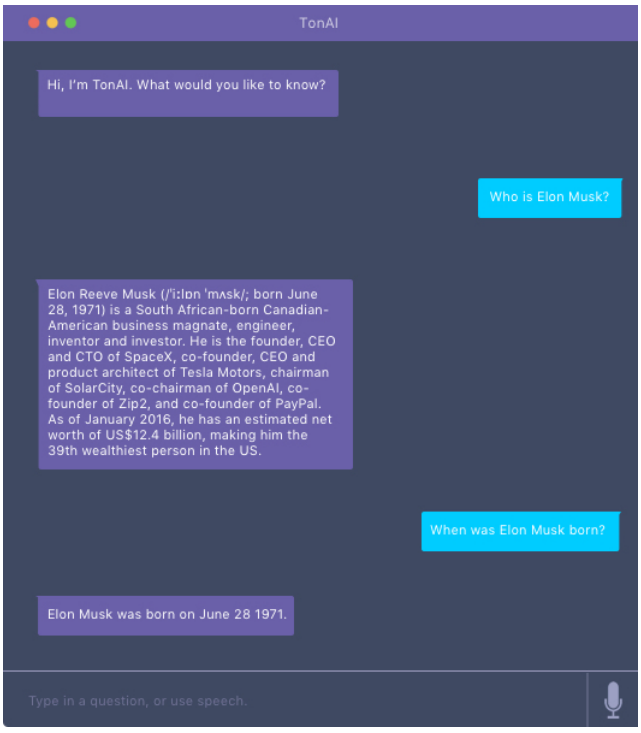


Figure 6 - GUI mock-up

3.3.4 Tools

The initial report mentioned a few tools which were potentially going to be used. After some testing and project management there are three main tools which are being used in this project.

NLTK (Natural Language Tool Kit)

NLTK [3] is an open source Python library used for natural language processing. This will specifically be used for the converting natural language into a SPARQL query, and to convert the information retrieved from DBpedia into a natural language answer. With over one hundred corpora and lexical resources NLTK is a superb tool to use in projects such as this Open Domain Question and Answer system. As development stands NLTK will be used for, tokenisation of a sentence, and POS tagging.

DBpedia

A crucial part of the system, DBpedia [4] is being used as a knowledgebase for the system to retrieve answers. It uses Wikipedia as the source of its information, and using semantic web practices, and Linked Data, such as the resource description framework (RDF), DBpedia allows the data to be organised in a more machine readable manner. This makes it perfect for a project such as this. As mentioned previously no question and answer system can be truly open, yet. However, DBpedia contains information on 4.58million items. Realistically speaking the system will not be able to retrieve information on all 4.58million items, and there is no clear way of measuring how many items it will be able to access. With this said, with access to a vast number of items we can say the system will be able to provide answers to most general questions.

SPARQL & SPAQL Wrapper

To query DBpedia and retrieve information SPARQL must be used. The goal is to parse a question into a SPARQL query, then use the parsed query on DBpedia to retrieve an answer. A SPARQL query contains three key elements; a prefix, what can be thought of as a property, and the location of the data, in this case a URL to the relevant DBpedia page. Using the Python SPARQL wrapper [5], the system will first go access the DBpedia page, look for elements with the specific prefix, then find properties with those prefixes.

```
SELECT ?rdfs
WHERE {<http://dbpedia.org/resource/London>} rdfs: comment ?comment
```

Figure 7 - Example SPARQL query

Figure 7 shows an example SPARQL query. Where rdfs is the prefix, comment is the property, and of course the URL is “http://dbpedia.org/resource/London”. This query will retrieve a short paragraph comment on London from DBpedia.

3.4 Performance Requirements

The system will be tested mainly on Unix based systems such as Mac OS and Ubuntu, with some testing being completed on Windows. It should be able to run on most environments and on computers with a broad range of specifications. As it currently stands the computer running the system will need the SPARQL wrapper, and NLTK modules which are easily obtainable and free. A crucial aspect needed to use this system is an internet connection. Using DBpedia as the knowledgebase requires a constant connection to retrieve information, without it the system simply won’t retrieve any information, thus won’t be able to produce an answer. With the size of the information being between bytes and kilobytes a fast internet speed isn’t required. An internet speed of 5mbs would suffice.

One of the secondary objectives is to implement a voice API, so the user can ask the system a question through speech, and the system can reply using speech. Should this feature be implemented the system requirements would require a microphone and a speaker. If no microphone is present but a speaker is, the system will be able to output audio, but not input audio.

3.5 Flow Diagram

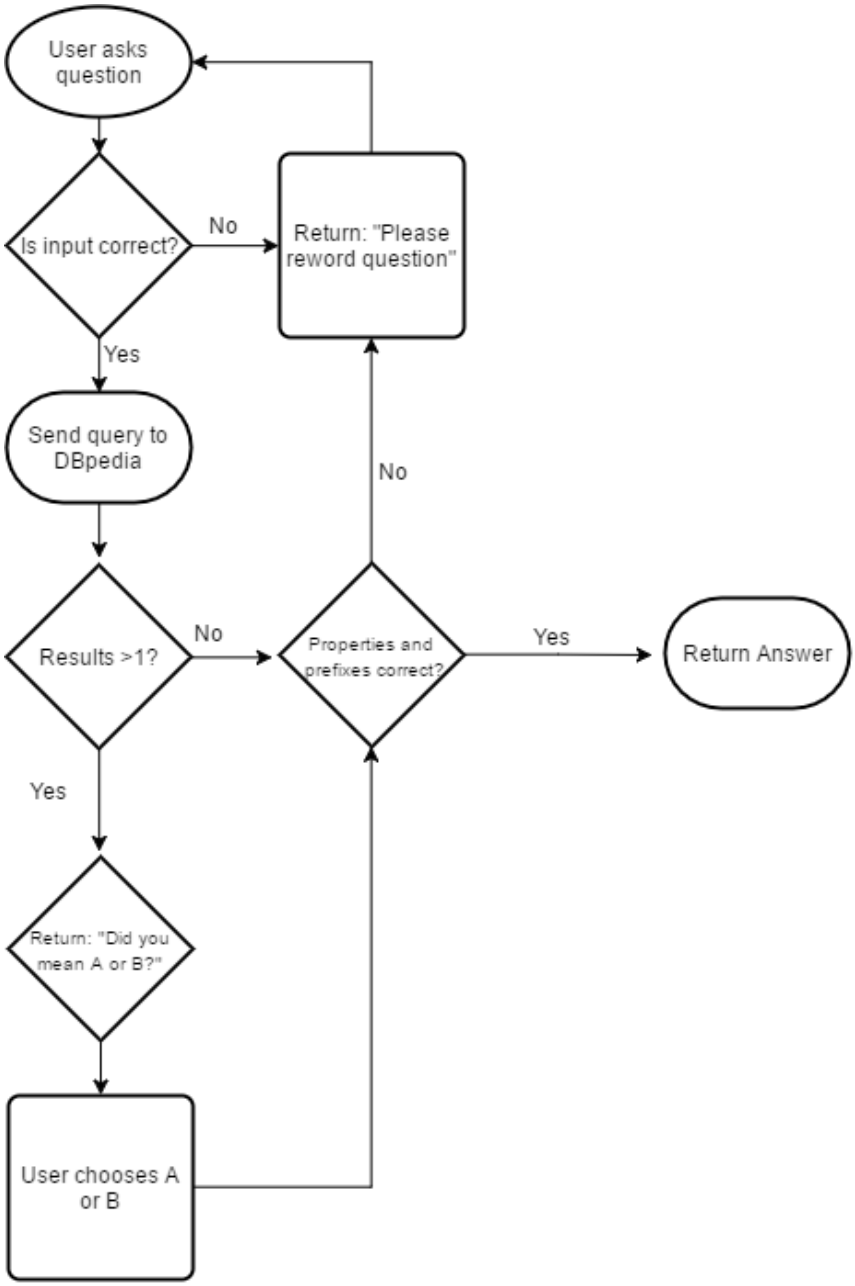


Figure 8- Flow Diagram

The above flow diagram gives a straightforward explanation of how the system works with informational retrieval. The first decision the system must make is if the input given is correct. Has the user input a question or something else? The parsed query is then sent to DBpedia, which will be used to refine the question if the search yields more than one result. After this DBpedia will return an error if the properties or prefixes are incorrect. If an error is returned the user will be prompted to reword the question so the natural language parser can redefine the POS tags, to properly query the database.

3.6 System Architecture Diagram

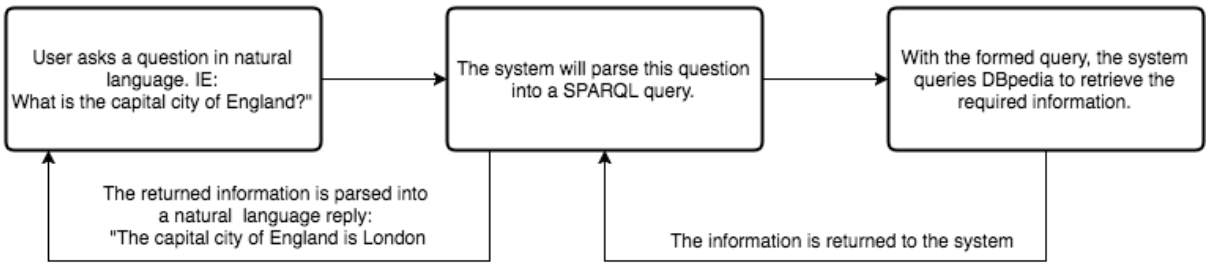


Figure 9 - System Architecture Diagram

The above diagram shows the architechtre of the system. It gives a brief overview on how the main parts of the system work with each other to produce an answer. The system is asked a question, it parses the question, turns it into a SPARQL query, queries DBpedia, returns the information to the system, then finally parses the information into a natural language reply.

3.7 Requirements

These are functional and non-function requirements of the system. The functional requirements specify what the system should do, whereas non-functional requirements describe how the system acts.

3.7.1 Functional Requirements

- Parse a natural language question into a SPARQL query
- Use the parsed SPARQL query to query DBpedia
- Parse the retrieved information into a natural language answer
- A user-friendly functioning GUI (Time permitting)
- Allow the user to ask a question using speech (Time permitting)
- The system should be able to handle incorrect input

3.7.2 Non-Functional Requirements

- Response time should be under 3 seconds
- The system should be intuitive
- Interaction with the system should be seamless
- No user data should be stored or collected
- The system should be reliable

4.0 Literature Review

4.1 Open Domain Question and Answer Systems

Open Domain Question and Answer systems are becoming increasingly prominent in modern day devices, and is an increasing area of interest in the field of artificial intelligence. They are now built into all major mobile devices in the form of a virtual assistants. Such assistants have also been ported into desktop operating systems, for example MacOS uses Siri [6], and Windows 10 has Cortana [7] which is also featured on the Xbox One. Amazon have developed the Amazon Echo, which features a virtual assistant named Alexa [8], Google have also just released Google Home [9] a competitor to the Echo.

With confidence, it can be assumed the next wave in consumer technology will have an emphasis on such artificial intelligence assistants, a crucial part of which would be their question and answer systems. Not only are such systems built into devices, but also standalone applications now have QA systems in the form of Chatbots. Facebook messenger has hundreds of such Chatbots [10], such as one from the Wall St Journal (WSJ), which when asked will return news headlines. The Wall St Journal Chatbot has a closed domain, as the system is used to only access articles and information the WSJ website. This not only provides more accuracy for results, but the fact an institution such as the WSJ shows how prominent QA systems are becoming.

For mass user adoption, it is crucial for a QA system to have two main things. Great natural language processing, and a vast domain (open or closed). Most systems are quite good at reading in natural language. However, parsing this and answering questions requires both great natural language and an in-depth domain so the system can answer a multitude of questions in many ways. When one fails, to the user, the entire system fails. Artificial intelligence must be seamless.

4.2 Information Retrieval Using the Semantic Web

This system will use semantic web methods like RDF to retrieve information. With QA systems, there are multiple ways of retrieving information from a source. One of which could be referring to list of documents and extracting keywords. This type of information retrieval would be apt for a more closed domain system, say a software troubleshooting system. The type of information retrieval needed for a system like the one being developed uses semantic web methods like RDF.

As reported by John W.T. Lee and Alex K.S. Wong, of the Hong Kong Polytechnic University [11], traditionally information retrieval on the web is be done using keyword search. The results of which may lack precision as often words have more than one meaning. The idea of the semantic web,

conceived by Tim Berners Lee, hopes to avoid this issue while making the web more machine readable. This highlights one of the key steps that need to be taken to enhance this area of artificial intelligence, making information easily readable and retrievable for machines. The more information a machine can read and retrieve, the larger it's domain will be. With a larger domain, a machine will be more capable of answering a variety of questions, thus becoming more open.

4.3 Natural Language Engineering

Natural language engineering, often referred to as natural language processing, is an integral field within artificial intelligence. The goal of this field is to essentially process natural language into a machine-readable format. Within the context of question and answer systems, such as the one being developed, natural language processing is being used to ask questions, and have them answered.

To read natural language a system needs to tokenize a sentence [12], break it down into an array of individual words, then assign POS grammar tags to the words. Once this process is completed the system will be able to analyse the question and create a relevant query.

The report *Natural language question answering: the view from here* by L. Hirschman and R. Gaizauskas [13], gives an intriguing overview on questions and answering using natural language. They outline methods to use for question analysis. One of which being putting constraints on the user input, this could be done by limiting vocabulary or syntax of a sentence. Remaining on the topic of question analysis they write about look for "key question words". These include words such as *where* which implies the user wants to know about a location, or *when* which implies the user wants to know of a time.

References

[1] O. Rahman, “CE301 Initial Report”.

[2] DBpedia, “Predefined Namespace Prefixes,” [Online]. Available: https://dbpedia.org/sparql?nsdecl.

[3] NLTK Project, “NLTK 3.0 documentation,” 2015. [Online]. Available: http://www.nltk.org/.

[4] DBpedia, “DBpedia,” [Online]. Available: http://wiki.dbpedia.org/.

[5] C. T. (. F. C. I. H. (. A. Z. Sergio Fernández (Redlink), “SPARQL Endpoint interface to Python (1.7.6),” [Online]. Available: http://rdflib.github.io/sparqlwrapper/.

[6] Apple, “Siri,” Apple, 2016. [Online]. Available: http://www.apple.com/uk/ios/siri/.

[7] Microsoft, “Meet Cortana,” 2016. [Online]. Available: https://www.microsoft.com/en-gb/windows/cortana.

[8] Amazon, “Echo,” 2016. [Online]. Available: http://www.amazon.co.uk/echo.

[9] Google, “Google Home,” 2016. [Online]. Available: https://madeby.google.com/home/.

[10] Facebook, “How To Build Bots For Messenger,” 2016. [Online]. Available: https://developers.facebook.com/blog/post/2016/04/12/bots-for-messenger/.

[11] A. K. W. John W.T. Lee, “Information Retrieval Based On Semantic Query on RDF Annotated Resources,” 2004.

[12] NLTK Project, “nltk.tokenize package,” [Online]. Available: http://www.nltk.org/api/nltk.tokenize.html?highlight=tokenize#module-nltk.tokenize.

[13] R. G. L. Hirschman, “Natural language question answering: the view from here,” Cambridge University Press, 2001.

Figures