Using Natural Language Processing to Map Character Interactions		
Shane Quinn	**C11759215**	**Mark Foley**

1. Project statement

The aim of this project is to create a program that will analyse the text of a novel, short story or play/screenplay using Natural Language Processing (NLP) techniques, such as Named Entity Recognition (NER) and Coreference Resolution. Using these techniques the program will learn to differentiate or recognise which words are the names or pronouns referring to specific characters. The program will then create a graph of the interactions between the various characters over the course of the source text.

2. Research

2.1 Background research

This section will explore the background research that forms the basis of this project. It will explain what is meant by Natural Language Processing, give a brief history of the field along with explanations of some of the key terms and tasks involved in the implementation of this project.

Natural Language Processing

Natural Language Processing (NLP) is an area of research, in Computer Science, Artificial Intelligence and Linguistics, that aims to create applications to analyse, understand and manipulate natural language text or speech. NLP uses a wide range of techniques and has numerous applications, though arguably the ultimate goal of NLP is to achieve artificial systems that can process language as humans do.

NLP covers a wide range of research areas and implementation tasks. Below are some of the most frequently researched areas of NLP.

- Machine Translation (MT) – MT is concerned with utilising computers to translate text from one human language to another. NLP originated from research in to MT and many NLP techniques were developed as part of research in to this area.

- Information Retrieval (IR) – IR is concerned with searching and retrieving stored data. While it is arguably more closely linked to databases, within the field of Computer Science, recent years have seen more IR systems developed using NLP techniques.

- Information Extraction (IE) – IE is concerned with extracting and classifying information in a structured way from an unstructured text. Information Extraction is the key focus of this project and contains many subtasks including, Named Entity Recognition, Coreference Resolution, Part-of-Speech Tagging and Chunking. These terms will be explained later in this section.

- Automatic Summarisation – The aim of auto summarisation is to analyse a document or other form of information and to return a summary of relevant and important content.

A Brief History of NLP

NLP has its origins in the late 1940s, when research began in to Machine Translation. While the idea of mechanical translators had been around for centuries, it was in the 1940s that Warren Weaver proposed the idea of using digital computers to translate documents between natural human languages. In 1949 he wrote a memorandum titled, 'Translation', that inspired a lot of research in to machine translation. Weaver had been involved in code breaking during World War II and he approached machine translation as though a

document written in one language could be viewed as having been written in code. As such if you could crack the code then you could translate the document in to another language.

Unfortunately for early researchers, language proved to be more complex than they had initially thought, so research teams turned to the area of Linguistics. In 1957 Noam Chomsky published his book Syntactic Structures. In the book he introduced the idea of generative grammar, which was an approach to the study of syntax and attempted to define sets of rules for syntactic structures. This fuelled further research into both linguistics and NLP ideas, such as machine translation and speech recognition.

However, progress was slow and in 1966 a report from the Automatic Language Processing Advisory Committee (ALPAC) concluded that machine translation was not achievable given current technologies and understanding. This led to less research in the area of machine translation being funded. In spite of this, this period saw some significant developments in theoretical work and systems. For example SHRDLU, a system, developed by Terry Winograd, that simulated a robot interacting with and manipulating blocks on a tabletop. While it was very limited, this system showed that it was possible for computer systems to process and understand language, at least on a small scale.

The next major break through in NLP, however, didn't come until the 1980s. Prior to the late 1980s NLP systems used complex sets of hand written rules, but the introduction of machine learning algorithms combined with the steady increase of computational power modernised the approaches to NLP.

Early machine learning algorithms used models such as decision trees to replace the hand written rules with if-then rules. However modern approaches have moved away from this symbolic approach to a statistical approach to analysing language.

Statistical approaches to NLP apply a number of statistical methods to a large text corpus in order to analyse common linguistic patterns. A corpus is a large collection of sample texts. By training NLP applications on the realistic grammar used in these texts NLP programs are better able to deal with ambiguity in a text.

Part-of-Speech Tagging

Part-of-Speech (POS) Tagging is the process of tagging and identifying words within a given sentence. POS Tagging will identify if a word is a noun, verb, adverb, adjective etc., based on the word itself along with its relationship to words in its vicinity.

One of the major issues in POS tagging is that of the ambiguity of language. If we take the following sentence as an example:

They refuse to permit us to obtain a refuse permit.

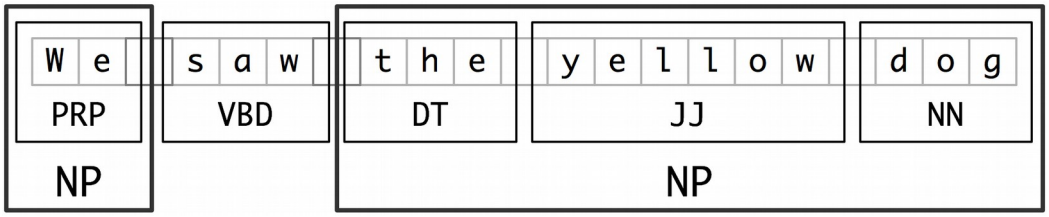
In this sentence the words refuse and permit appear twice. However in the first instance they appear as verbs and in the second instance they appear as nouns. (eg. In the first instance refUSE is a verb that means to deny, while the second instance it is the noun REFuse meaning rubbish).

POS taggers use the structure of the sentence in order to infer the correct tag to use in ambiguous case like this.

Chunking

Chunking is a form of parsing that involves segmenting a sentence in to phrases so that related words are members of the same phrase.

The figure below illustrates a sentence that has been chunked in to two separate noun phrases.



Sentence segmented in to Noun Phrases

The first noun phrase contains the pronoun 'We' and the second contains the phrase 'the yellow dog'. Here we can see that each noun phrase collects information in relation to a single noun entity. 'We' refers to the group of people looking at the 'dog'. We can also see from the second noun phrase that there is a description of the dog as yellow.

By chunking sentences in these noun phrases we can detect entities within the text as well as gaining further information about these entities.

Named Entity Recognition

Named Entity Recognition (NER) is a natural language processing technique. It is a subtask of information extraction and seeks to identify and locate named entities in a given text. While an exact definition of what constitutes a named entity is hard to pin down, the Conference on Computational Natural Language Learning (CoNLL-2002) provides the following definition.

"Named entities are phrases that contain the names of persons, organizations, locations, times and quantities."

If we take the following sentence as an example:

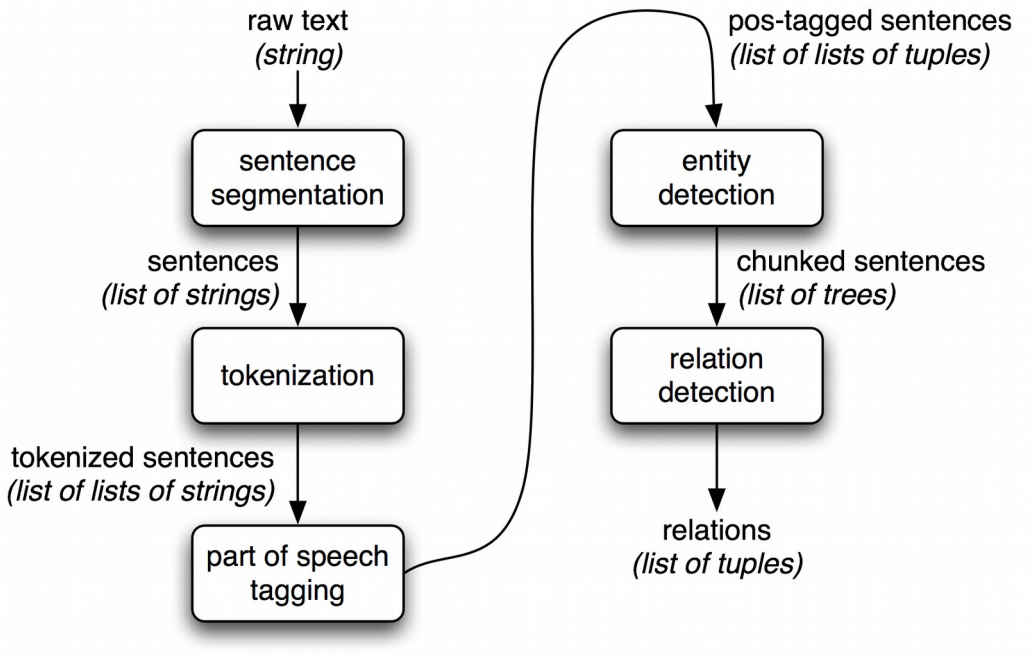
Jim bought 300 shares of Microsoft in 2006.

A NER program should reproduce the sentence with annotations that highlight and classify the named entities, as seen below.

$[\text{Jim}]_{\text{person}}$ bought 300 shares of $[\text{Microsoft}]_{\text{organisation}}$ in $[2006]_{\text{time}}$.

The figure below illustrates the steps involved in detecting named entities.

Figure 2:



Architecture of named entity and relationship detection

Firstly the text is segmented in to sentences. The sentences are then tokenised in to words. The words are then tagged with an appropriate tag, i.e. noun verb etc. The tagged sentences are then chunked in to noun phrases and categorised. Lastly relationships can be detected from these noun phrases.

While there is no standardised method for evaluating NER systems, both CoNLL, the Conference on Natural Language Learning and MUC, the Message Understanding Conference, use an F_1 measurement to evaluate the effectiveness of an NER system.

An F-measure is the harmonic mean between precision and recall. This is represented as follows:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Therefore,

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The MUC-7 conferences defined precision and recall as follows:

Precision = COR/POS
Recall = COR/ACT

COR is the number of entities correctly tagged.
POS is the number of possible entities, i.e. the number of entities tagged by the system.

ACT is the number of actual entities within the test data.

The two human annotators at MUC-7 achieved F_1 scores of 97.60% and 96.95%, while the best system at MUC-7 scored 93.39%, thereby achieving near-human performance.

Coreference Resolution

Coreference Resolution is a subtask of Information Extraction that seeks to identify all expressions in a text that refer to the same entity. This involves find references to the entity that may not use the entity's name. If we take the following two sentences as an example:

Jack visited Jill’s house. He made her dinner.

In this example the words 'Jack' and 'He' refer to the same entity. As do the words 'Jill' and 'her'.

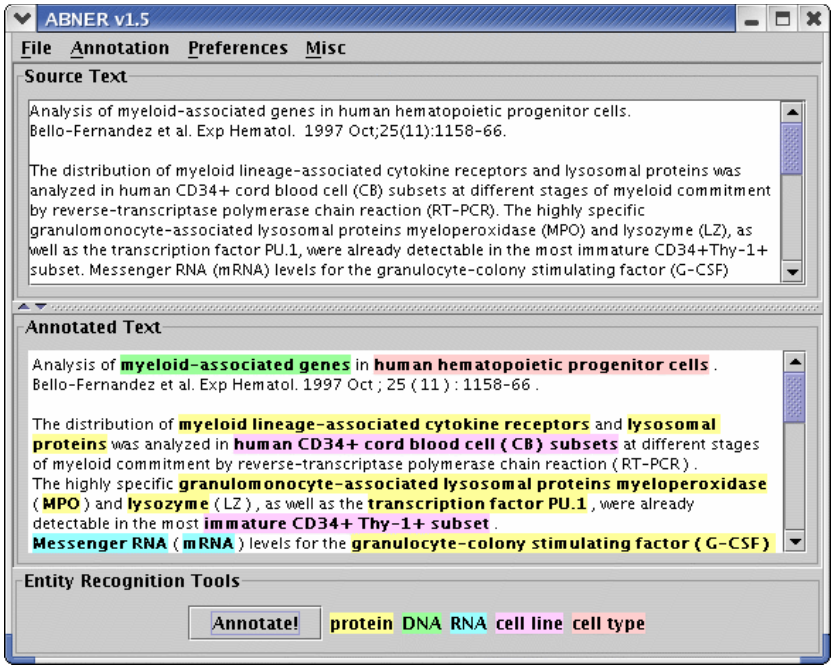
2.2 Alternative and Similar Systems

This section will look at two systems that are similar to my project, namely ABNER and Citeology. It will provide a brief introduction to, and an evaluation of, each of these systems.

ABNER

ABNER, which stands for A Biomedical Named Entity Recogniser, is a Java based, open source NER tool designed to analyse molecular biology texts. Originally developed in 2004 by Burr Settles at the University of Wisconsin-Madison part of the NLPBA/BioNLP 2004 Shared Task challenge, its current version, 1.5, was released in 2005.

ABNER consists of a simple Java GUI and uses a statistical machine learning approach to identify and annotate named entities such as proteins, DNA, RNA and cell types. It is bundled as a Java archive (JAR) and includes a Java API to allow it be incorporated in to other applications.



ABNER screenshot

Evaluation

Usability: ABNER's simple interface is very easy to use. It has two text boxes, one for the source text and another for the annotated text. It also features four drop down menus at the top of the window.

File has options for starting annotation of a new file, opening a text file containing source text to annotated and the option to save annotated text in one of three formats.

The annotation menu has two options. The first is to annotate the current document, which can also be done by means of an 'Annotate' button at the bottom of the interface. The second option is to perform a batch annotation of all files stored in a directory. These files can be output in the same three formats available in the save function.

The preferences menu allows users to choose from 2 different annotation models. The first is NLPBA, which will annotate 5 different classes of entities, protein, DNA, RNA, cell line and cell type. The second is BioCreative which will annotate proteins only.

The final menu is misc. This menu allows for one of three different demo texts to be loaded in to the source box. This menu also contains an 'About ABNER' option, which provides version and licensing information along with a link to the ABNER website.

Along side the ability to load from a file and generate demo text, text can also be manually entered in to the source text box.

Clarity of Data: ABNER presents the annotated data in a very clear way. The five different entities are colour coded in the annotated text box. A legend presented at the bottom of the interface lists the 5 entities highlighted by the colour representing them.

Documentation: There is no documentation for ABNER. The website lists the features available and provides the screenshot shown above. However the simple interface and ease of use easily counteracts the lack of documentation.

F_1 Measure: ABNER has two F_1 scores for each model. It has a standard F_1 score and soft F_1 score. Given length and complexity of the named entities it is tagging the soft F_1 score allows for matching one boundary of the entity with a one-word error on the other side being tolerated.

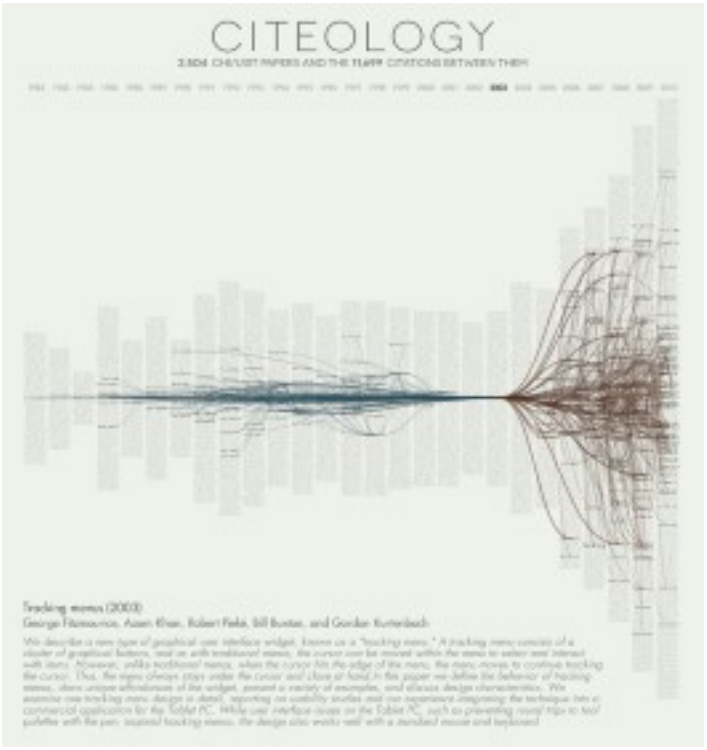
NLPBA F_1: 70.50% Soft F_1: 82.00%
BioCreative F_1: 69.90% Soft F_1: 83.70%

Given the complexity of named entities in molecular biology texts, these F_1 scores are considered roughly state of the art.

Citeology

Citeology is an interactive visualisation tool to show the relationship between research papers based on their citations. Citeology was developed as a Java applet by Justin Matejka, George Fitzmaurice and Tovi Grossman at Autodesk Research, as part of Autodesk's Visualization Project.

Citeology runs as a Java applet on the Citeology webpage and allows users to select one of the 3502 papers from both the ACM Conference on Human Factors in Computing Systems (CHI) and ACM Symposium on User Interface Software and Technology (UIST) conferences, between the years 1982 and 2010. The applet then shows a graph connecting all the older papers referenced in the selected paper along with all the papers since that have referenced the selected paper.



Citeology Screenshot

Evaluation

Usability: Citeology is quite easy to use. There are 27 columns on screen each with a year on top. These columns contain the titles of all the papers published at the CHI and USIT conferences for the that year. Hovering over a title will display the paper's full title in a tooltip box. Clicking on a paper will select it and will show the paper's ancestors (paper that it has referenced) and descendants (papers that reference it). The ancestors a connected to the paper with blue lines while the descendants are connected using red lines.

There are buttons in the top left hand corner to set the number of generations of descendants and ancestors to view. First generation ancestors and descendants are papers directly referenced by, or that directly reference, the selected paper. Further generations of ancestors and descendants are paper referenced by ancestors or papers that reference descendants.

There is also a 'Find Paper' search box in the top left corner. This allows the user to directly search for a specific paper and will begin to automatically suggest paper titles based on letters typed by the user.

There are three buttons at the bottom of the interface. The first is the 'Visit Paper Page' button which links users to the selected paper on the ACM's digital library website. The next button allows the user to generate a high-resolution pdf of the current graph. The third button allows the user to copy a link to the specific graph they have generated.

Below these three buttons the title, authors and beginning of the selected paper are displayed.

Clarity of Data: The major drawback of Citeology is the lack of a zoom function. The data is too difficult to read in the applet itself. The titles of each paper are impossible to read without the tooltip popup.

The graphing itself is well done and while showing three or more generations often leads to a mass of lines, this in itself demonstrates the interdependency of these paper and how they build upon each other. The ability to show one, then two and more generations in a sequence is visually impressive but due to the lack of clarity a user would have to generate a pdf to show for each iteration in order to clearly see the differences.

Documentation: The interface is quite easy and intuitive to use. Citeology does however provide documentation on how to use the applet, including explaining keyboard shortcuts for certain task. This is visible just below the applet.

2.3 Technologies researched

This section will introduce a variety of NLP toolkits that are available, compare and contrast these toolkits and explore which technologies are most suitable for the project. The section will also cover the work completed in order to familiarise myself with these technologies.

Toolkits Researched

There a variety of toolkits and programming libraries available, under various different licenses, for performing common NLP tasks. I researched a number of different technologies in order to decide which would best suit the project.
The criteria for evaluating the NLP toolkits looked at the language used, the license, the tasks it performed, and the documentation available.

LingPipe

LingPipe is a suite of natural language processing tools for Java designed by Alias-i.

Language: Java

License: 3 levels of paid license, 1 free license Affero GPL(AGPL)

Tasks Available: A large number of common NLP tools including, Classification, Named-Entity Recognition, Part-of-Speech Tagging, Coreference Resolution, Sentence Segmentation, Sentiment Analysis and Language Identification.

Documentation: Javadoc containing all classes, tutorials available on the website and a textbook (price: $40.49).

NLTK

The Natural Language Toolkit (NLTK) is an NLP toolkit for building natural language processing programs in Python. It was created in 2001 as part of a computational linguistics course in the Department of Computer and Information Science at the University of Pennsylvania.

Language: Python

License: Apache License, Version 2.0

Tasks Available: A large range of NLP tools including, Named-Entity Recognition, Part-of-Speech Tagging, Sentence Segmentation, Chunking and a large number of tools for analysing the content of text.

Documentation: Full API documentation, a wiki available on the NLTK GitHub page and a free online textbook.

OpenNLP

OpenNLP is a Java machine learning toolkit for natural language processing.

Language: Java

License: Apache License, Version 2.0

Tasks Available: Sentence Detection, Part-of-Speech Tagging, Named Entity Recognition, Classification, Chunking and Coreference Resolution

Documentation: Javadoc containing all classes, a wiki and a free online manual

Stanford CoreNLP

Stanford NER is a collection of NLP tools developed at Stanford.

Language: Java

License: GNU General Public License

Tasks Available: Part-of-Speech Tagging, Named Entity Recognition, Coreference Resolution and Sentiment Analysis.

Documentation: Javadoc containing all classes and limited documentation available on the website

Comparison

The criteria chosen to evaluate these four toolkits were the language used, the license that the toolkit was available under, the NLP tasks it could perform and the documentation available for the toolkit.

The language used was important for a number of reasons. Firstly, it needed to be a language that I already knew or one that could be learned with relative ease. This was due to the time constraints of the project and the fact that I would already have to learn NLP techniques to complete the project.

The license that the toolkit was available under was chosen as the project has a no budget. Therefore, if the toolkit was not free, or relatively cheap, then I would be unable to use it.

The NLP tasks that could be performed by the toolkit was chosen as a criterion because a more powerful toolkit meant less time spent attempting to create implementations of these tasks from scratch.

Finally the documentation available was quite important as it would mean less time spent learning how to use the toolkit and therefore more time researching, designing and implementing the project.

While the Stanford CoreNLP certainly contained all the necessary tools in order to complete the project, the documentation available is extremely limited. The tutorials available for LingPipe are quite good, however the expense of having to buy the book was a barrier to gaining access to all documentation available. OpenNLP provides a free online manual and NLTK has a free textbook. Of the two, I found that while the OpenNLP manual was quite good, the NLTK textbook was a wealth of information. While both provide examples on using the toolkits in question, the NLTK book is a great introduction to the Python, NLP and the toolkit itself.

In terms of the tasks that the toolkits were able to perform all four were mostly even. The NLTK and LingPipe had the most functionality, but all four covered most of the tasks needed to complete the project. NLTK was the only one of the four toolkits that did not have built in functionality to perform Coreference Resolution. It is, however, possible to build a Coreference Resolution module for the NLTK and one has already been contributed to the NLTK github repository.

The last criterion was the language used. Three of the toolkits use Java, with only the NLTK using Python. For writing NLP programs I have yet to find anything that conclusively shows that one of these languages is better than the other. Each language has its own pro's and con's, however none of these specifically relate to NLP programs. Java programs tend to be more portable and faster at run time, while Python programs tend to much simpler and cleaner to write and debug. In terms of writing NLP programs my research simply suggested to write in my preferred language.

Based on the evaluation criteria I have chosen to use the Python based NLTK. This is due to the relative ease of learning Python, the wealth of documentation available for the NLTK and the wide range of NLP tasks it can perform.

Learning New Technologies

As part of this project I have had to learn a new programming language, Python, along with the Python based NLTK.

I began studying Python first, as I would need it in order to learn how to use the NLTK. The NLTK textbook does contain an introduction to Python, but it is mostly based on areas that will be needed to use the toolkit. There are a lot of resources available online for learning Python, for example, learnpython.org, codeacademy.com and wiki.python.org.

learnpython.org and codeacademy.com both start with the basics, such as variables and basic operators, and gradually work up to more complex topics, while wiki.python.org has a huge selection of links to various books, websites and interactive tutorials for both those who have no programming experience and those who do.

I used these tools briefly in the beginning to get used to the syntax. However, once I had a grasp of the syntax, I set myself the challenge of redoing a selection of programming labs that I had been given in my first and second year, while learning C and C# respectively. This work greatly improved my understanding of the language and its capabilities.

I then began to study the NLTK. The NLTK textbook has twelve chapters, containing introductions to NLP techniques, examples, definitions and a range of exercises in each chapter. I completed the first few chapters quite quickly, as I had familiarised myself with Python already. The rest of the chapters I went through carefully and completed as many of the exercises as time allowed. This helped me greatly in understanding the toolkit and it's uses.

2.4 Findings

This section will explore the findings of and the system requirements identified as a result of the research and analysis carried out.

Functional Requirements

Through research, the implementation of a number of NLP tasks are key to the functionality of the project. These are sentence segmentation, part-of-speech tagging, chunking, named entity recognition and coreference resolution. These task alone however will not make a complete system. The proposed system will also need to import text, either in a raw format or with the ability to convert to a raw format.

The system will also be required to create graphs based on the result from the NLP analysis. The graphs should represent the data in a clear manner. Lower priority, optional graphing functionality may include the ability to represent the data in differing levels, such as the ability to select graphs for individual or multiple selected characters as well as individual chapters, scenes or acts.

The program should hold the data from the NLP analysis in a simple database, to allow for ease of repeated viewings of graphs from novels or plays that have already been analysed.

Functional Requirements Matrix

Priorities in this matrix are represented as follows:

H High Priority
M Medium Priority
L Low Priority

Req ID	Name of Req	Description	Priority
1	Import Text	Import text from a file	H
2	Convert Text	Convert text from other formats for use in the program	L
3	Sentence Segmentation	Perform sentence detection and segmentation	H
4	POS Tagging	Perform part-of-speech tagging on the segmented sentences	H

5	Chunking	Perform chunking on tagged sentences to separate them in to noun and verb phrases	H
6	NER	Perform named entity recognition to identify characters in the text	H
7	Coreference Resolution	Perform coreference resolution to identify all references to each character.	H
8	Write to database	Ability to write information to a database	M
9	Read from database	Ability to read information from a database	M
10	Graphing	Ability to graph the results of the NLP analysis	M
11	Grouping Graphs	Ability to group the graphed data in a variety of ways	L

Non-Functional Requirements

Based on research in to similar systems there are two key non-functional requirements for the proposed system.

Usability: The system should both be easy to use and easy to learn. Given that user interaction with the proposed system is limited, ease of use should be simple too achieve.

Clarity of Data: A key element of the proposed system is to represent data in a graphical format. It is important that the information should be presented in a clear, relevant manner and should be easily interpreted.

2.5 Bibliography

Alias-i.com, (2014). *LingPipe Home*. [online] Available at: http://alias-i.com/lingpipe/index.html [Accessed 11 Dec. 2014].

Autodeskresearch.com, (2014). *Citeology - Projects - Autodesk Research*. [online] Available at: http://www.autodeskresearch.com/projects/citeology [Accessed 11 Dec. 2014].

Bird, S., Klein, E. and Loper, E. (2014). *NLTK Book*. [online] Nltk.org. Available at: http://www.nltk.org/book/ [Accessed 11 Dec. 2014].

Clark, A., Fox, C. and Lappin, S. (2010). *The handbook of computational linguistics and natural language processing*. Chichester, West Sussex: Wiley-Blackwell.

Cs.bham.ac.uk, (2014). *SEM1A5 - Part 1 - A brief history of NLP*. [online] Available at: http://www.cs.bham.ac.uk/~pjh/sem1a5/pt1/pt1_history.html [Accessed 11 Dec. 2014].

Itl.nist.gov, (2014). *SAIC Information Extraction*. [online] Available at: http://www.itl.nist.gov/iaui/894.02/related_projects/muc/ [Accessed 11 Dec. 2014].

Lewis, D. and Jones, K. (1996). Natural language processing for information retrieval. *Commun. ACM*, 39(1), pp.92-101.

Matejka, J., Grossman, T. and Fitzmaurice, G. (2012). Citeology. *Proceedings of the 2012 ACM annual conference extended abstracts on Human Factors in Computing Systems Extended Abstracts - CHI EA '12*.

Mitkov, R. (2003). *The Oxford handbook of computational linguistics*. Oxford: Oxford University Press, pp.219-222.

Nlp.stanford.edu, (2014). *The Stanford NLP (Natural Language Processing) Group*. [online] Available at: http://nlp.stanford.edu/software/corenlp.shtml [Accessed 11 Dec. 2014].

Settles, B. (2005). ABNER: an open source tool for automatically tagging genes, proteins and other entity names in text. *Bioinformatics*, 21(14), pp.3191-3192.

Settles, B. (2014). *ABNER: A Biomedical Named Entity Recognizer*. [online] Pages.cs.wisc.edu. Available at: http://pages.cs.wisc.edu/~bsettles/abner/ [Accessed 11 Dec. 2014].

Soon, W., Ng, H. and Lim, D. (2001). A Machine Learning Approach to Coreference Resolution of Noun Phrases. *Computational Linguistics*, 27(4), pp.521-544.

Surface.syr.edu, (2014). [online] Available at: http://surface.syr.edu/cgi/viewcontent.cgi?article=1043&context=istpub [Accessed 11 Dec. 2014].

Webknox.com, (2014). *Named Entity Definition*. [online] Available at: http://webknox.com/p/named-entity-definition [Accessed 11 Dec. 2014].

Www-nlpir.nist.gov, (2014). *MUC 7 Proceedings*. [online] Available at: http://www-nlpir.nist.gov/related_projects/muc/proceedings/muc_7_toc.html#named [Accessed 11 Dec. 2014].

3. Analysis

This section will describe in detail exactly what the project should achieve.

The proposed system should do the following:

Text Input

The program should have a number of pre-existing texts for the user to choose from, along with the ability to import text from a file. This will be limited to raw text initially, with functionality for importing text from plays, novels and short stories from .pdf files to be developed, depending on the time constraints of the project.

Analysis

This is the core of the project. The program will analyse the text and perform and number of NLP tasks. Firstly it will segment the sentences, in order to begin tagging. It will then perform Part-of-Speech (POS) tagging of the sentences. Next it will chunk the sentences in noun phrases. It will then perform Named Entity Recognition to locate and identify the characters within the source text. Lastly it will perform Coreference Resolution, to identify pronouns and which characters they represent. A possible lower priority functionality would be to identify groups of characters who belong to a single organisation or group.

Database

The results of the analysis above will stored in a simple database.

Graphing

Lastly the program will use the information stored in the database to plot a graph of the interactions of these characters over the course of the text. The graph should have a number of functions, including the ability to show an individual character or groups of characters and the ability to show different time frames, for example certain chapters of a novel or scenes/acts within a play.

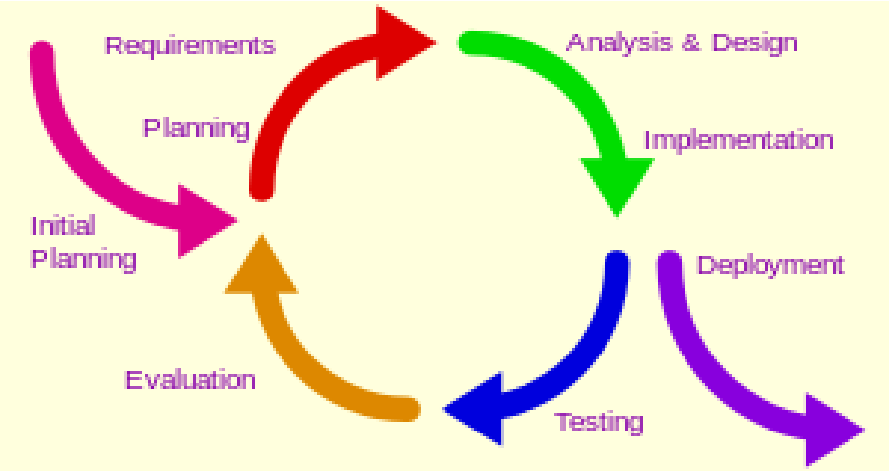
Interface

The program will initially be developed with a command line interface. A simple GUI may be constructed, depending on time constraints.

4. Approach and Methodology

This section will cover the intended approach to implementing the project and the reasoning behind the choice of approach.

While I will not necessarily be sticking to a strict methodology for this project, I have decided to take an iterative and incremental approach. The basic idea of this approach is to develop small portions of the project through repeat development cycles. As represented in the figure below each cycle contains several parts, and the cycles can be repeated as needed until the project is ready for deployment.



Incremental and iterative cycle

After initial planning has been completed, each development cycle will involve four key steps. Firstly establishing the requirements for the functionality that will be built in this cycle, if this is not the first iteration there may be some changes to the requirements based on lessons learned in the previous cycle. This will then be followed analysis and design or redesign and implementation. The functionality that has been implemented will then be tested. The results of the testing will then be evaluated and the results of this evaluation will affect the next cycle. For example if all tests have passed, the next cycle may involve developing new functionality. If the testing fails however the next cycle will involve a re-evaluation of the requirements, design or implementation in the next cycle.

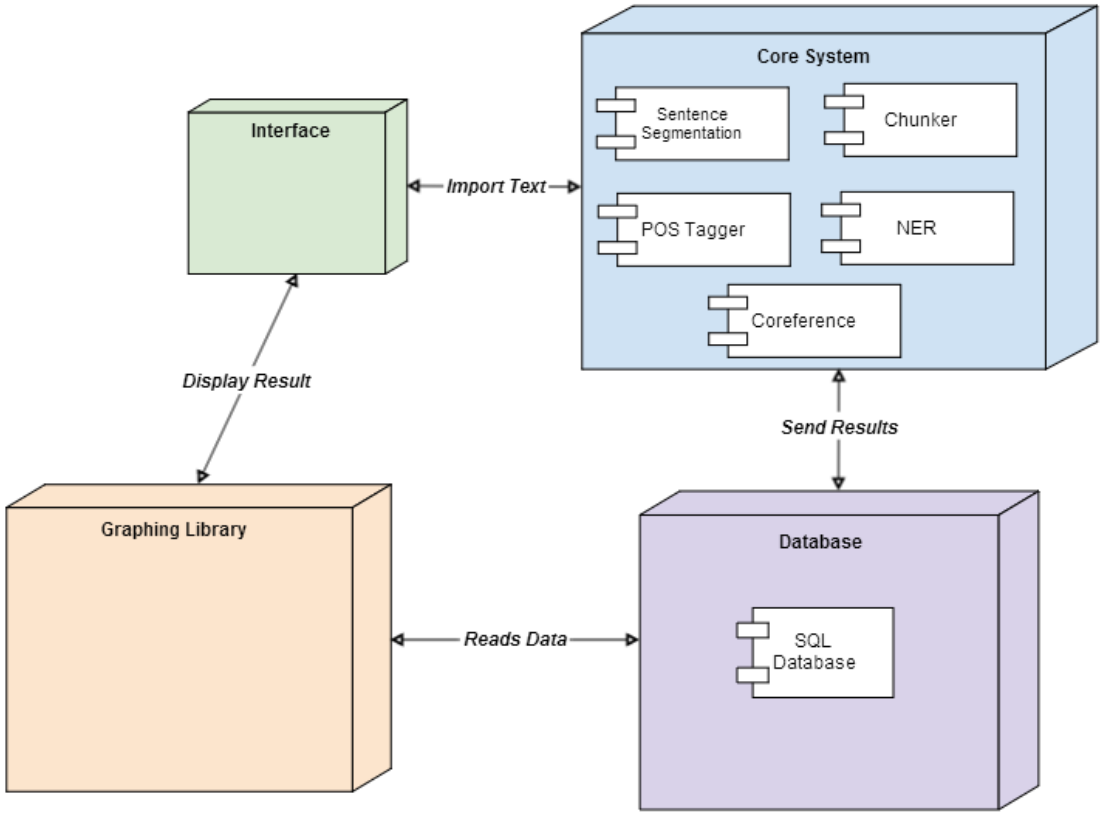
I have chosen to approach the project in this way for a few reasons. Firstly, the technologies and techniques involved in delivering this project are still quite new to me and as such it is possible that the requirements or design of the project may need to be tweaked or changed as the project is implemented. Secondly, I have identified a number of lower priority features that will be implemented towards the end of the project, provided that there is time to do so.

By adopting this approach, I am giving myself both the ability to develop and test the key components of this project, within the time constraints, while still allowing myself the flexibility to respond easily to changes that may need to be made or complications that may arise, over the course of the project.

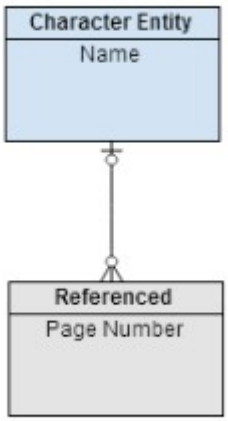
5. Design

This section contain the design diagrams.

5.1 Technical Architecture Diagram



5.2 Entity Relationship Diagram



6. Prototyping and Development

This section will describe the prototyping and development work completed to date.

While no concrete development has begun to date, some work has been completed in terms of prototyping. It was necessary to do some minor coding while learning to use the NLTK. As such, some basic functionality has been prototyped.

The elements prototyped to date included sentence segmentation and part-of-speech tagging. Also prototyped is some basic named entity recognition. The named entity recognition prototyping is quite basic and has not yet be trained on basic sentence structures, but is still able to recognise entities to a degree.

7. Testing

This section will cover the planned testing strategy for the project, identify stakeholders involved in testing and outline some planned test cases.

Testing Approach

As outlined in the methodology section, development will occur in a number of iterative development cycles, with testing at the end of each cycle. Testing during development will take the form of dynamic unit testing involving test cases. The test cases will be designed to test the particular functionality that was implemented during the current development cycle, with integration testing to be carried out at the end of cycles were individual modules are combined.

When the interface has been developed, whether it is command line based or a graphical user interface, the system will be tested for ease of use. This will be involve a combination of test cases and observing both computer literate users and average users as they interact with the system.

Finally, the system's F_1 measurement will have to be evaluated in order to gauge the efficacy of the system's named entity recognition functionality.

Test Cases

Below are two early test cases to test the system's ability to import text and to detect and segment sentences.

Test Case Number: 1
Test Case Name: Import Text
Purpose: To test the system's ability to import text from a file
Procedure Steps: **1.** Select 'Import Text' **2.** Select file to import **3.** Check test output to see if the text from the file has been imported
Expected Result: The file will import text from the file in to the system

Test Case Number: 2
Test Case Name: Sentence Segmentation
Purpose: To test the system's ability to detect and segment sentences from a block of text

Procedure Steps: **1.** Import text **2.** Run sentence segmentation code **3.** Save output **4.** Inspect output
Expected Result: The system will return a list of individual sentences

8. Issues and risks

This section will explore the main challenges still faced by the project and the approach that will be taken to solve them.

There are a few challenges facing completion of this project in the areas of coreference resolution and named entity recognition.

While most named entity recognition should be relatively straight forward and coreference resolution should handle pronouns quite well, dealing with aliases for characters will prove more difficult. Often in stories, be it novels or plays, a single character may be referenced as several different names. An example from recent times is the antagonist of the Harry Potter series who is referenced as his birth name Tom Riddle, his chosen name Voldemort and among his enemies as 'You-Know-Who' and 'He-Who-Must-Not-Be-Named'. These are four different names that refer to one character.

A more global approach to coreference resolution and named entity recognition will have to taken to solve this problem. Along with the standard analysis, which will focus on individual segmented sentences, a wider analysis must also be done. This may not necessarily have to focus on attempting to analyse the story as a whole for these references. Often there will be an explicit reference, possibly in surrounding sentences that reveal a character's aliases.

Another challenge will be that of the time constraints of the project. I have identified a number of low priority features that will add to the project if implemented, but will not affect the feasibility of the project if they are not.

9. Plan

This section will explore the key deliverables needed to complete the project.

Key Deliverables

The key deliverables of the project are as follows:

- Complete implementation of the basic functionality of the project. Specifically, importing text, sentence segmentation, POS tagging and chunking.
- Complete implementation of named entity recognition.
- Complete implementation of coreference resolution.
- Complete implementation of graphing functionality.
- Submit Final Report and demonstrate completed program.

There are of course more deliverables than the ones mentioned above, but these a the key thing that need to happen in order for the project to be a success.

10. Conclusions

As seen in this report to date I have completed research in to NLP and a number of new technologies that I have familiarised myself with. The key functional requirements have been identified and there is a basic design in place for the system. I have chosen an approach and explained my chosen approach and testing strategy.

While there may be some issues and risks associated with the project I am confident that they will be overcome. I believe this project is feasible and has the potential to be an impressive project.