A Web-Based Text Editor for a Controlled Natural Language

Udit Hari Mehta

Bachelor of Engineering
Software Engineering Major

MACQUARIE
University
SYDNEY·AUSTRALIA

Department of Electronic Engineering
Macquarie University

November 13, 2017

Supervisor: Dr. Rolf Schwitter

ACKNOWLEDGMENTS

First of all, I would like to thank my parents for the unconditional love and support they have always given me. Without them, I would be nowhere.

I would like to express my most heartfelt appreciation to Dr. Rolf Schwitter for providing me with his support, expertise, and guidance throughout this semester and project. His calm demeanour and expertise from the start of the project gave me a motivation to put my best foot forward and helped me feel at ease. Without his guidance, I would not have been able to have a great learning experience.

At last, I would like to thank my friends, who, away from home, have always kept me going.

STATEMENT OF CANDIDATE

I, Udit Hari Mehta, declare that this report, submitted as part of the requirement for the award of Bachelor of Engineering in the Department of Engineering, Macquarie University, is entirely my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualification or assessment at any academic institution.

Student's Name: Udit Hari Mehta

Student's Signature: Udit Hari Mehta (Software)

Date: November 13, 2017

ABSTRACT

In recent years, there has been massive improvement in predicting what a user wants to type as advanced technology has been more accessible than before. User interface, and human-computer interaction have become vital in today's world. This has encouraged a project that explores the user-interface of a text editor, of the Processable English (PENG), which is a machine-oriented controlled natural language (CNL), which before this, was not explored the way it deserved. A CNL limits the natural language (base language) by putting constraints over syntax, semantics and lexicon.

A speech interface API, to provide speech input, was implemented in the previous prototype, which had functionality problems. This paper presents the change in user-interface, by making it more appealing and easier to use. Additionally, the entire system (code) was documented to better understand the system while improving the human-computer interaction. The problem where the system would not allow multiple sentences to be entered in the text editor of the PENG system was also aimed to rectify.

Contents

1. Introduction 1

1.1 Project Scope 1

2. Background and related work 3

2.1 Types of CNL 3

2.1.1 Human oriented CNLs 3

2.1.2 Machine Oriented CNLs 3

2.1.2.1 Attempto Controlled English 4

2.1.2.2 PENG^{ASP} 4

2.1.2.3 Computer Processable Language 5

2.1.2.4 Evaluation of ACE, PENGASP and CPL 5

3. Literature Review of PENG 8

3.1 Structure of PENG 8

3.2 Predictive Text editor 8

3.3 HTTP Server 9

3.4 Working of the CNL Processor 9

3.5 Existing PENG Projects 9

3.5.1 Limitations of the current PENG system 11

3.5.2 Solving the Limitations 11

3.6 Conclusion 12

4.Timeline 14

5. Requirements 16

5.1 User Requirements 16

5.2 System Requirements 17

5.2.1 Functional Requirements 17

5.2.2 Non-Functional Requirements 17

6. Tools required and Implementation 18

6.1 Model-View-ViewModel 18

6.2 Single Page Application 19

6.3 Git and GitHub 20

6.4 OneDrive 20

6.5 JavaScript and jQuery 20

6.6 KnockoutJS 20

6.7 Superfish 21

6.8 SWI-Prolog 21

6.9 Predictive Text Editor Implementation 21

6..9.1 Input Area 21
6.9.2 Menu Interface 23
6.10 Programming Methodology 23
6.10.1 Iterative prototyping 24
6.10.2 Basis of Programming Logic 24
7. Working of Peng 26
7.1 Current State of Project 26
7.2 Working Setup 27
7.3 Demonstration of Interface Interaction 27
7.3.1 Typed Input 27
7.3.2 Menu-based Interface 29
7.3.3 Using the prediction to enter specifications 30
7.4 Saving, Loading, and Adding Demonstrations 31
7.4.1 Saving Specifications 31
7.4.2 Loading Specifications 32
7.4.3 Adding Token to Lexicon 34
8. Discussion and Current Work 37
8.1 Understanding the code and Documentation 37
8.2 User Interface design 37
9. Explanation of important JavaScript files 50
9.1 KeyEventHelper.js 50
9.2 ViewModel.js 56
10. Risk Assessment 74
10.1 Loss of data, Damaged Machine or Stolen Machine 74
10.2 Time Management 74
10.3 Not Understanding Requirements 74
11. Conclusion 76
12. Future work 78
13. Abbreviations 80
14. References 82
Appendix A 84
Appendix B 85

Chapter 1

1. Introduction

Natural language processing [1] has given impetus in understanding and processing languages which are natural. Subsets of natural languages that are acquired by limiting the vocabulary and grammar are called Controlled Natural Languages (CNLs) [5]. With the increase in computing power and processing, there has been great advances in analysing, understanding, and generating languages that are natural. This advancement has led to a better understanding of the human language and breaking it down into structures that can be easily understood and implemented. But natural languages are ambiguous and can be interpreted in more than one way. This however can be improved with the use of the CNLs [2].

CNLs [2] enables to keep a respectable level of a language's natural properties while getting rid of ambiguity (or minimizing ambiguity). A CNL is a subset of a written natural language (base language) which uses restricted grammar such as lexicon, syntax, and semantics [2]. A CNL can be written in such a way that it holds most of the nature of the base language, so the users of the base language are able to interpret and use the CNL completely [2]. CNLs are divided into 2 groups: human-oriented CNL and machine-oriented CNL [4] which are explained in the literature review.

1.1 Project Scope

The project focuses on a machine-oriented English-based Controlled Natural Language (CNL) by exploring and extending an existing text editor to CNL processor. The Processable English (PENG) editor is machine-oriented CNL [4] which is built for knowledge representation [12]. The PENG system has been designed in such a way that the words/sentences written in PENG are easily understood by humans while also making it easy for the machine to process it. Differentiating from other CNLs, the user does not need to understand or remember the rules on which it is built and the PENG editor guides and helps the user using predictive interface techniques [13]. The current PENG implementation provides for text input and speech input. PENG editor is not portable onto any web browser, except, Google Chrome, this is the challenge that needs to be overcome in this project. The current version of the PENG editor uses Google Speech API which makes the speech input of the PENG editor incompatible on all web browsers else Google Chrome. As a result, the project will drop the idea of having speech as an input. At the moment, there are no sophisticated text editors that guide the writing process of a CNL in an unobstructed way. Therefore, it would be beneficial to create such a tool that helps a user to write efficiently in a CNL. The outcome of this research has broader applications, for example: a CNL could also be used as a high-level interface language to a database or any other knowledge system.

Chapter 2

2. Background and related work

As introduces earlier CNLs mimic natural languages by limiting it's (natural language's) vocabulary and grammar [5]. This is done to get rid of or limit complexity and ambiguity of full natural language. It is to note that a CNL must have only one natural language as its foundation. Let us take the following English sentence as an example: "I see a girl on the mountain with a telescope" is an ambiguous one, which means the sentence can be interpreted in more than one way leading to confusion and complexity. It can mean the following:

A: I have the telescope and I see a girl on the mountain through it.

B: I see a girl who is on the mountain and she has a telescope.

From the two interpretations above, it becomes clear that CNLs are required to eliminate ambiguity. A CNL allows to reconstruct ambiguous sentences into unambiguous ones. CNLs can be labelled into the following two groups: 1. Ones that help improve readability (e.g. for non-native speakers), these languages are called simplified or technical languages (also known as human oriented controlled languages) [4] [6], and 2. Those that warrant definite and valid automatic semantic analysis of the language, these are machine-oriented controlled natural languages [4, 7].

In Section … we go into broader details of the above-mentioned types of CNLs. while Section 3.4 is about the architecture of the PENG. In Section…, limitations of the current edition of the PENG editor are put forward and ways to overcome it which is followed by a conclusion in Section….

2.1 Types of CNL

2.1.1 Human oriented CNLs

When there is a need to enhance the quality of the technical documentation, human=oriented CNLs are used [4], [6]. They help to simplify the machine translation of documents. These languages constraint what the user can write by using general rules, for example (Keep sentences short, Use only passive voice, etc) [4], [6].

2.1.2 Machine Oriented CNLs

For machine-oriented CNLs a formal syntax and semantic is defined, and can be drawn to formal language which already exists (such as first-order logic or answer set programming) [4, 7]. Therefore, these CNLs are casted as high-level knowledge representation languages, and writing of those languages is aided by redundancy checks fully automatic consistency checks and, query answering, etc [4, 7].

PENG is a machine oriented CNL and uses a Definite Clause Grammar notation and a chart parser to translate the language into an answer set program. Other machine-oriented CNLs that have been discussed and compared by Dr. Rolf Schwitter [11] are Attempto Controlled English (ACE) and Computer Processable Language (CPL). Also called, general-purpose CNLs as they were formed without any distinct application domain in consideration.

These languages are created in a way so that automatic translation into a formal target language is possible and be used for automated reasoning.

The following section will talk about machine-oriented languages and then there will be an evaluation of the discussed machine-oriented languages.

2.1.2.1 Attempto Controlled English

Attempto Controlled English (ACE) had been created as to give the domain experts with a tool that would provide better knowledge representation in English language without its ambiguity. This would make it better to understand and easier to write (17). ACE uses first order logic that helps to automatically translate the ambiguous ACE text into an unambiguous one. This process is carried out by the Attempto Parsing Engine (APE). ACE supports a wide range of vocabulary. Ranging from high order clauses to anaphoric expressions. It also takes care of noun phrases, plurals, and questions. If someone use and interacts with the ACE words or text, one can conclude that they can be intuitive and natural and tend to be understood quite easily by the user. This maybe the case because most of the controlled languages are developed with thought of making it easy for human to comprehend.

Interestingly, there is some difference from most of the controlled languages. There is learning needed to type in ACE text as it supported by an editor, and this would help the user to write words or texts that is understood or allowed by the language (as certain word or text may be restricted to avoid ambiguity) [11,17,18].

2.1.2.2 PENG$_{ASP}$

The PENG$_{ASP}$ system is a machine-oriented language which [9] is an authoring tool that guides the writing process of the user and helps to write non-monotonic specifications and then these specifications are translated into a modelling language that then helps to solve complex problems. This modelling language is known as Answer Set Program (ASP). ASP can be thought as declarative programming language, and knowledge-intensive applications make use of ASP [19].

The definite clause grammar (DCG) is the principle on which the language processor is based. Then it is processed using a chart parser. The result or the target language is ASP which makes it different to ACE [20].

One of the main advantages of the PENG system, is that it does not require the user to remember the underlying structure of the controlled language that it uses, which is a similarity that it shares with ACE. The specifications that are entered in the editor of the PENG system, are sent to the server as a token, or at token level. With each word in the specification having a unique id. This helps to provide the predictive functionality of the PENG system which helps to guide the writing process of the user. Thus, by using this a guide, user gets to know what is accepted and recognized by the system and enforces the constraints on the grammar of the language on the go [11]. The PENG system is discussed in detail in Section 3 and Section 4.

2.1.2.3 Computer Processable Language

Another language came into being that converts sentences of English into a knowledge representation language that is based on base language and has certain restrictions [21]. This language which was created at Boeing Research and Technology came to be known as Computer Processable Language (CPL). Like in many other controlled natural languages, CPL has an interpreter that works out ambiguity at different levels which is based on rules defined for the processing of the language. CPL is different to ACE and PENG. It's different to ACE and the PENG.

ACE uses a relatively small set of rules for the interpreter, whereas PENG uses a predictive text editor to guide the user while typing, but the CPL has texts that are differentiated into three categories, these are questions, ground facts, and rules. The sentences in the CPL then take the following structure:

grounds facts takes three forms [11]:

"These is|are NP"

"NP verb [NP] [PP]*"

"NP is|are passive-verb [by NP] [PP]*"

Out of the five forms of questions, the following two are the main ones: "What is NP?"

"Is it true that sentence?"

As far as rules are concerned, the following pattern (or form) is accepted by the CPL

"IF sentence [AND Sentence]* THEN Sentence [AND Sentence]*

2.1.2.4 Evaluation of ACE, PENG$_{ASP}$ and CPL

The editor used by ACE, helps to simplify the writing process by giving users/writers a text editor that could be used to type ACE specifications. A bit similarly, PENG makes use of a predictive text editor that, as mentioned earlier, guides the writing process of the user and makes it easier to write PENG specifications, where CPL takes of ambiguous sentences and specifications when it carries out language processing. These three languages, like many other controlled natural languages, try and succeed to make the writing process easier. There interface makes grasping the context easier as well.

. . . .

These three languages possess the tools that supports or guides the user while they are interacting with the respective language and writing specifications in that language. But still, these require some added functionalities or restrictions in the controlled language so that they can be used more extensively, because, at the moment only those sentences, that are syntactically correct according to the restrictions are processed and allowed to type.

These languages stand out as the users does not have to worry about the underlying principles and structure of the controlled language and learning does not require much effort. This is in complete contrast to Basic English, that gives the user absolutely no tool to help or aid in the writing process.

The PENG translates the PENG text to ASP, whereas CPL convers the specifications to formal KR, and the ACE takes the ACE text and then processes it into first-order-logic, which makes machine translation possible.

Chapter 3

3. Literature Review of PENG

Introduction

PENG is a machine-oriented controlled natural language written in DCG notation and processed by a chart parser [9]. A predictive text editor supports the writing process and this made it the first CNL that offered sophisticated writing support. When a user enters a word into the editor, it generates a list of options as look ahead information to inform the user as to how they can proceed with the sentence [9]. As there are syntactic restrictions, this process ensures that the grammar rules of the CNL are being followed by the user. The resulting specification can then be translated unambiguously into an answer set program. The language processor of PENG is based on DCGs [8]. Hence, it is important to understand what DCGs are. A DCG is a general grammar format which allows us to specify the structure of sentences [8]. DCG are so named because they show/define a grammar as set of definite clauses written in Horn clause logic [16].

Abstract

The abstract has already been provided at the start of this paper.

3.1 Structure of PENG

In this section, we are going to talk about the predictive editor of the PENG system and how it communicates with the HTTP Server using JSON objects [9].

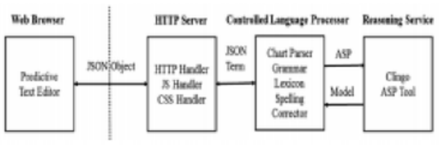


Figure 1: Architecture of the PENG System

3.2 Predictive Text editor

The predictive editor is at the heart of the PENG system. There is asynchronous communication between the predictive editor and the CNL processor which then translates the specification text into Answer Set Programs (ASP) that can be executed and used for consistency checking and question answering [9]. The CNL processor generates look-ahead categories through which the user can choose the next word and provides anaphoric expressions which the user can select instead of typing them again.

The PENG^{ASP} system uses a client-server architecture. In a client-server architecture, a client (computer) requests for resources that are given by the server [9]. The predictive text editor works in a web browser, in our case Google Chrome and communicates via an HTTP server with the CNL processor. An ASP tool is used as reasoning service by the CNL processor.

3.3 HTTP Server

The HTTP server asynchronously communicates with the predictive text editor using JSON objects [9]. JavaScript and jQuery are used to implement the predictive text editor where the pull-down menu functionality is provided by the Superfish plug-in , whereas the web server is implemented in SWI Prolog along with the language processor [9]. Prolog, a logic programming language, is chosen because it is easy to implement a unification-based grammar for processing the CNL. Using the URL http://localhost:8085/peng/, the web browser is connected to the HTTP server that uses certain JavaScript, HTML and CSS handlers load the predictive editor while establishing the communication session between the editor and the CNL processor [9].

3.4 Working of the CNL Processor

PENG translates a specification into an extended discourse representation structure (eDRS) [9] and supports the following features:

•Incremental parsing is done during the writing process [9]. At the same time, natural language processing takes in which anaphoric expressions are taken care of and then an eDRS is generated [9]. After the generation of an eDRS, a paragraph is generated after which look-ahead information is resolved and generated.

3.5 Existing PENG Projects

As known, this project is using an existing prototype and trying to extend it by improving its user interface and functionality errors. We hope to rectify those issues in this implementation.

Below are the screenshots of three previous implementations

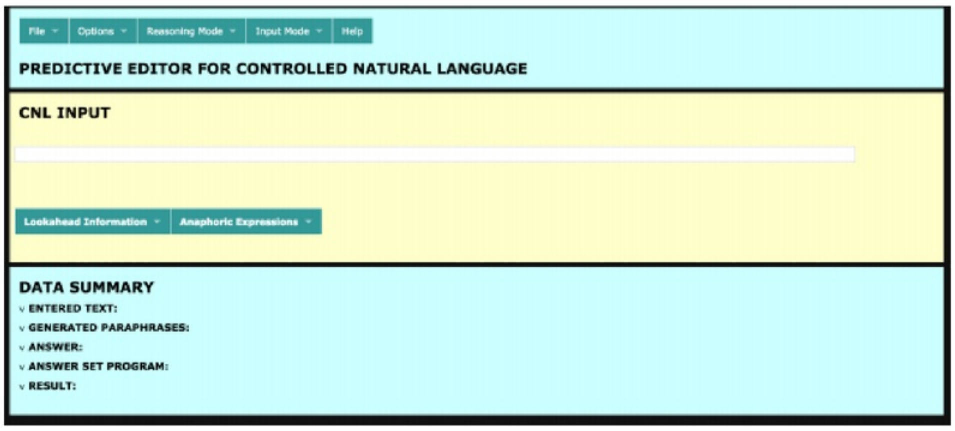


Figure 2: Guy's Implementation

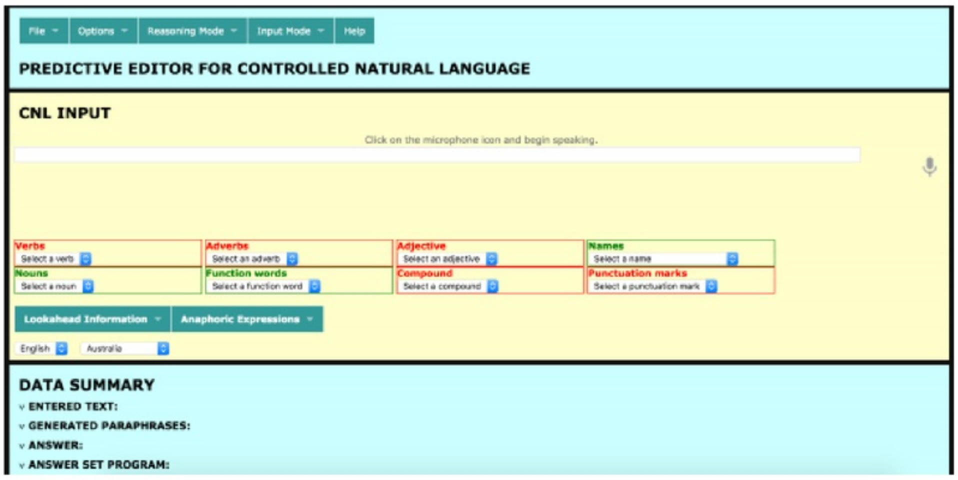
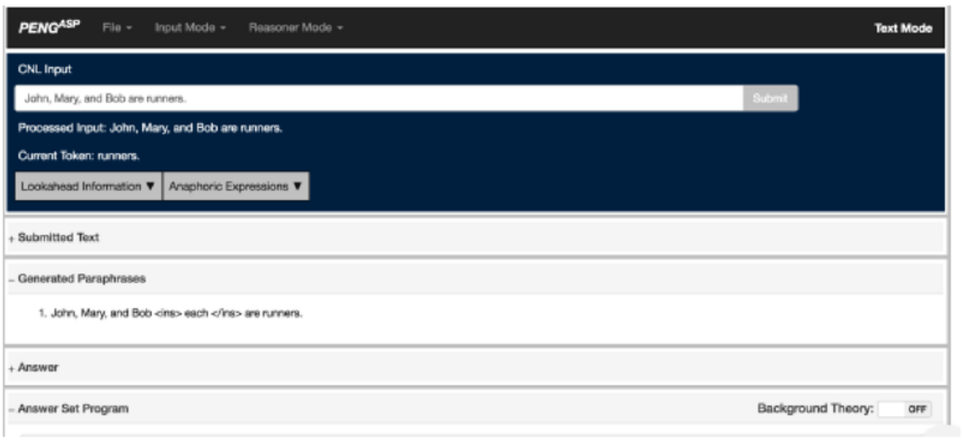


Figure 3: Nalbandian's Implementation



CNL Input

John, Mary, and Bob are runners.

Processed Input: John, Mary, and Bob are runners.

Current Token: runners.

1. John, Mary, and Bob <ins> each </ins> are runners.

Figure 4: Bernardez's Implementation

3.5.1 Limitations of the current PENG system

The last screenshot, which is Bernardez's implementation, is the latest implementation and the basis of this project. Following are the limitations of the implementation that have been identified and aimed to rectify.

1. The first and major limitation is the portability of the PENG system. It works well on the Google Chrome web browser but when it is used on other web browsers there is limited functionality. For example, if used on Firefox the user interface is not optimized. When a word is entered there is a problem in the lookup pull down menu. (As we would expect of a prototype, there are still some bugs).

2. Multiple sentences cannot be typed or entered in the text editor. When a sentence is typed, and it finished with the full stop (.), one cannot modify that sentence. This is a flaw as one should be able to type multiple sentences and modify existing ones.

3. Once typed, the user cannot go back and change the sentence.

4. The speech input type does not work smoothly. One has to say the word extremely slowly for the text editor to process it.

3.5.2 Solving the Limitations

Limitation 1 can be solved by examining where the functionality differs in different web browsers and using reverse engineering to come out with a common API that can be used in all the web browsers.

Limitation 2 and 3 seem to be programming issues, where the text editor is implemented to not take multiple sentences and thus also cannot go back and change the sentences. As far as Limitation 4 is concerned, it would be better to completely drop the idea of incorporating the speech input type. As the current version uses Google's Speech API, it is difficult for it to run in any web browser apart from Google Chrome.

Another solution is to have the speech processing at the client-side using pocket Sphinx (or a simpler speech recognition engine) [10]. Another solution is to integrate multiple speech interfaces and then let the browser decide which one is compatible. But this will be the last consideration, as other limitations need to be rectified first.

3.6 Conclusion

There have been improvements that are made in the working of CNLS, however, there is still room for improvements. As far as the current PENG system is concerned, it needs to be made more sophisticated and easier simultaneously so that the user can interact efficiently with the PENG editor. The issue of cross-browser portability needs to be solved as one can't ask the user to run PENG editor in a specific web browser.
Hence, it will be a massive step forward if the limitations described above are rectified. This will lead to better human-computer interaction and efficient processing of data.

Chapter 4

4.Timeline

The project has been divided into seven tasks and each task will have an allocated start time and an expected end time.

The tasks are as follows:

1. Understanding the current system

2. Refactoring existing code and collecting requirements

3. Deciding whether to build upon the current system or to make a text editor from scratch. system

4. User interface design

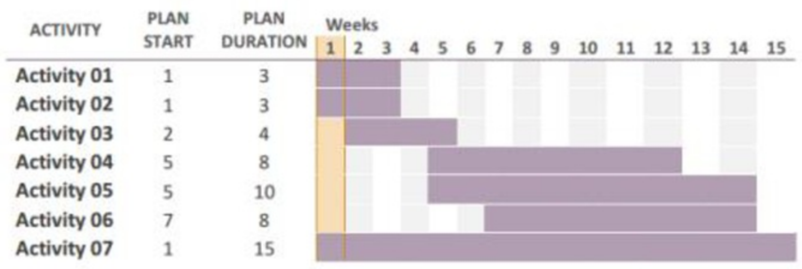
5. Implementing the code and maintaining the code

6. Evaluation of the system

7. Report writing

The following Gantt chart represents the timeline for the above activities:

31-07-2017 to 24-11-2017



ACTIVITY	PLAN START	PLAN DURATION
Activity 01	1	3
Activity 02	1	3
Activity 03	2	4
Activity 04	5	8
Activity 05	5	10
Activity 06	7	8
Activity 07	1	15

Figure 5: Timeline

Chapter 5

5. Requirements

This chapter will describe the software system to developed by defining its user requirements and system requirements.

The requirements were gathered using two methods: the first was to look at the existing systems defined requirements, and the second was to consult Dr Schwitter, who was acting as product owner, like those typically found in software projects. The overall system requirements were identical to the existing prototype with regards to functionality; however, the overall user interface design will be open for changes.

5.1 User Requirements

Central to the project are user requirements, hence these requirements are going to be discussed first. These requirements itemize what and how the user wants the system to look and behave.

User itself is the one who specifies the user requirements. In this project, Dr Schwitter, who is the stakeholder, has defined user requirements through meetings. Following are the user requirements that have been specified:

- User Requirement 1: The user is able to interact with the system without knowing its working
- User Requirement 2: The user can enter words and sentences without having to know the syntax and semantics of the system
- User Requirement 3: The user is able to update (delete, edit) characters, words and entire sentence of the text by using backspace key.
- User Requirement 4: The user is able to input multiple sentences.
- User Requirement 5: The user is able to see the result of the processed input immediately.
- User Requirement 6: The user is able to load and save specifications/puzzles.
- User Requirement 7: The user is able see lookahead information while typing.
- User Requirement 8: The user is able to modify existing sentences.
- User Requirement 9: The user is able to add unrecognized words to the lexicon.

5.2 System Requirements

The system requirements form the basis of the implementation of the code. These are grouped into two categories. These two categories are: Functional Requirements and Non-Functional Requirements. The following describe the functional requirements:

5.2.1 Functional Requirements

This section covers functional requirements which specify how the system should behave and what is required in order to get the system to behave in the specified way. These relate to what the input and its corresponding output should be. These requirements are complemented by non-functional requirements which are discussed in the next section.

• Functional Requirement 1: The system must work on any PC/Laptop/Notebook while running any operating system.

•Functional Requirement 2: The system must work on any web browser and not just Google Chrome.

• Functional Requirement 3: The menu interface must be updated with correct information when a token is added or removed.

• Functional Requirement 4: The system must allow the lexicon to accept a new token when the submitted token is unrecognized.

• Functional Requirement 5: The system must be able to save and load specifications.

• Functional Requirement 6: The system shall have one mode of operation: text mode while the speech mode needs to be removed.

• Functional Requirement 7: The system must display the correct answer set programs when a new specification is submitted.

• Functional Requirement 8: The system shall provide a suggestion box to assist the user in completing a word or sentence and it must be updated as soon as a new letter is typed.

• Functional Requirement 9: The system shall update the anaphoric expressions table when a new token has been submitted.

5.2.2 Non-Functional Requirements

These requirements complement the functional requirements by determining the quality of the system. Hence, these are also known as quality requirements. System attributes such as speed and time-complexity (performance), ease of maintainability, ease of availability, cost-effectiveness, time-effectiveness and load resistance are defined along with criteria or parameters. To test the non-functional requirements effectively and efficiently, one usually needs to wait till the end of the project, or in other words, until the system has been deployed. As the final result of the project will be a prototype, results of the both the implementation and non-functional requirements can be focused upon. The results of the non-functional requirements were not focused upon in the previous prototypes.

Chapter 6

6. Tools required and Implementation

The following sections will discuss the different tools and programming languages used in implementing the PENG system. As it's a complex system, the solution needs to be well documented. Knowing the application is an extremely responsive one, i.e., working at full capacity whether it's a web browser on a laptop or mobile phone. Hence, any drawbacks that arise from continuous communication between the client and the server, and the rendering of the server-side need to be taken care of. Therefore, choice of programming languages, tools and implementation tools to make sure these drawbacks do not arise.
A table containing the full list of tools used can be found in Appendix B.

6.1 Model-View-ViewModel

Most of the work done in this project is at the client end. Hence, an architectural pattern had been chosen to help further the development process. The architectural pattern chosen for the PENG system is the Model-View-ViewModel(MVVM), which is derived from the Model-ViewController(MVC).
MVVM [22] has three components: Model, View, and View-Model. Model refers to the way the system is structured. View refers to the part that the user sees and interacts with. View-Model refers to the position of the data in the model. View-Model acts as the bridge between View and Model while simultaneously isolating the two. In our PENG system, HTML file relates to the view, ViewModel written in JS is the view-model and objects are the models
As all the components are responsible for different roles, efficiency is increased.

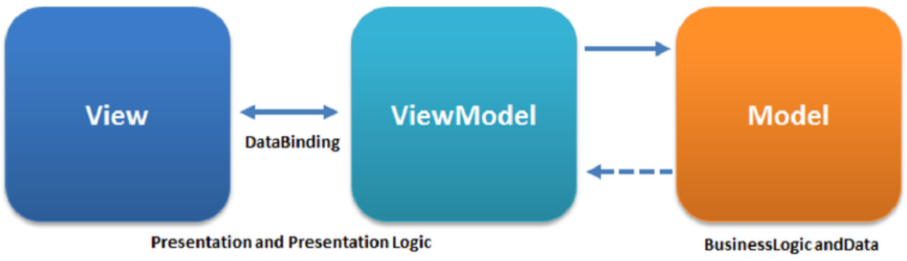


Figure 6: Model-View-ViewModel

6.2 Single Page Application

In our project we use a single HTML file that contains all the required components to make sure our user interface works. This is called A Single Page Application (SPA). Formally, SPA is a web-based application that uses a single HTML to connect all the components. Let's look at following code to understand it better.

```
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>PENG Editor</title>

    <!-- Required to use AJAX and JQuery -->
    <script src="../js_library/jquery.min.js" type="text/javascript"></script>
    <script src="../js_library/jquery-ui.js" type="text/javascript"></script>
    <link rel="stylesheet" type="text/css"
href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.17/themes/base/jquery-ui.css">


    <!-- knockout library for data binding-->
    <script src="../js_library/knockout-min.js" type="text/javascript"></script>


         <!—Required to use styles-->

    <link rel="stylesheet" href="../css/styles.css">
    <link rel="stylesheet" href="../css/mic-styles.css">

</html>
```

The above code is a part of the html code used in our project. The <script> tags and the <link> tags used, corresponding to the the JavaScript and CSS files that are used for interactivity and designing (styling) respectively).

User interface is updated dynamically with the use of SPA as it does not require the page to be loaded continuously. Using SPA has lots of advantages. It has the tendency to provide flawless user interface experience as web applications can be made responsive to ones own preference, which is not easily achieved when using applications across different platforms. SPA works brilliantly on highly responsive applications as an intermediate layer handles the communication between presentation and data. Using SPA, rendering can also be done on the client side instead of the server side [23].

6.3 Git and GitHub

The version control GitHub was used for this project to mitigate against the loss of data or damage to data. Different versions and changes were pushed onto the repository for review and to track changes. In case there is a version which has lots of bugs in it, GitHub would help to revert back to the state which had no or fewer bugs [24,25].
As it provides free services and it has the biggest community support, choosing GitHub was a no brainer. Apart from this GitHub provides great support for collaborating and sharing code. The use of GitHub should be mandatory for any code-related project as it provides safety against loss and corruption of data and lets one know about the progress that is being made.

6.4 OneDrive

Along with Git and GitHub, OneDrive was also used as changes can be saved on the fly and across different platforms.

6.5 JavaScript and jQuery

As it is a software project, the core of the project revolves around JavaScript and JQuery. JavaScript, is a programming language that is typed dynamically. JavaScript is designed for HTML and the web. It was chosen for the project because it is accepted everywhere as programming language of the web and supported by all web browsers in the market today. Along with JavaScript, JQuery [27] is used to simplify the implementation as it makes use of an already implemented library of code. It provides features such as DOM Manipulation, which stands for **Document Object Model** and it is a means to represent and interact with HTML files or documents. It provides event (which means to perform functions when a particular event is triggered), and AJAX . AJAX (asynchronous HTTP), is a jQuery function that is used to interact with the server and is used and required throughout the project.

6.6 KnockoutJS

KnockoutJS[26] is a JavaScript library that allows the developers to create their own interfaces that are in line with the corresponding data object model(DOM), which further enables DOM manipulation. Hence, using KnockoutJS MVVM architectural pattern. Using this, we have a control over DOM as it provides specific control in data which helps to scale the page. This increases computational efficiency as the things that are not related to the current scenario are never computed and prohibits recomputation of every single data item. . This is done possible because it has a system of observables, which figures out the relevant data and concentrates on computing just that relevant data.
Following are the features of the Knockout implementation

Knockout includes the following features:

- Declarative bindings
- The UI updates automatically when there is a change in the state of the data
- Helps to track dependency

- Provides a template that can be used, altered and added to suit and match the functionalities (such as jquery.tmpl).

It's a library, like ReactJS, but unlike AngularJS, which is a framework. This helps to reduce implementation complexity as it lets the application to gel with the components in the project.

6.7 Superfish

Superfish [28]is a JQuery Plugin that has been used to implement the menu interface for the predictive text editor. It strives to decrease the complexity that is involved when functionalities are implemented like event handling and expanding and collapsing of the menu.

Previous versions of the PENG system (system developed by Guy [9] and Schwitter [9] , and Bernardez) have used Superfish , which were highly responsive as far as these types of applications are concerned.

6.8 SWI-Prolog

SWI-Prolog has been used to implement the backend of this project/application. The backend comprises of the implementation of the server and the language processor. When the client-side and the language processor interact, they use the server as the medium. Here, a JSON object is taken from the client-side and is given to the language processor. It is then that the language processor looks and processes the input to check whether or not the input is in line with the constraints of the PENG language. If it conforms with the PENG language, then the lookahead data is returned along with spelling suggestion and anaphoric expressions, else it returns an error message.

The backend had been implemented already and as stated earlier, only the client-side needed changes which were to add handlers for the front-end files (HTML, CSS, and JavaScript files).

6.9 Predictive Text Editor Implementation

This section goes through the implementation of the predictive text editor. It's the biggest part of the project as a lot of functionalities and interactions are involved as per the requirements of the system. There are two crucial parts of the predictive text editor : Predictive menu-based interface and the input area.

6..9.1 Input Area

The input area had been implemented in a way that detects which character has been entered by the user and whether or not its conforms to the PENG language and ready to be sent as a token to the server. The following code refers to the way this was implemented

```
keyUpdate: function(d, e) {
        var keyID = e.keyCode == 13 ? e.keyCode : e.charCode;
        console.log(keyID)
        var keyVal = (String.fromCharCode(keyID)); // Character form
        var charAllowed = (keyID != 13 &&
viewModel.token().charAt(viewModel.token().length-1) != "." &&
viewModel.token().charAt(viewModel.token().length-1) != "?") || keyID == 13;

enterKey: function() {
        var isEndOfSentence =
viewModel.textAreaStr().charAt(viewModel.textAreaStr().length-1) == "." ||
viewModel.textAreaStr().charAt(viewModel.textAreaStr().length-1) == "?";

  Line 1      if(isEndOfSentence) {
            var submitStr = viewModel.textAreaStr().replace(" .", ".");
            submitStr = submitStr.replace(" ?", "?");
            textLineData.addSentence(submitStr);
            viewModel.textList.push(submitStr);
            viewModel.textAreaStr('');
            viewModel.token('');
            viewModel.$text_field.val('');
            viewModel.init();
        }
```

From the input parameters to the function keyUpdate(), it can be figured out that this is an event handler that would respond whenever a character is input. Line 1 above of the method checks if token has finished and we are at the end of the word and it is ready to be sent as the token. Using special characters to detect the end of the token was a smart idea. If and when a token has been completed, it would be sent to the server to be processed and then the predictive text editor would be updated with the data that had been fetched from the server. Else, a string variable is updated that is needed to keep tab on the current token.

The function below, postToken(word), is needed to check the token (the words that have been enter as specification) and the sent to the server. Not just this, it is also required to show the result in the Answer Set Program and Answer divs in the user interface.

```
    postToken(word) {
        // If word in lookahead
        var request = $.when(this.postToken2(word));
        request.done(function(data)
        {
            var json = JSON.parse(data);
            if (word == "." || word == "?") {
                viewModel.setAsp(json);
                if(word == "?") {
                    viewModel.setAnswer(json.answer);
                }
```

Any variable that gets updated or called is declared as an observable (mentioned in KnockOutJS [27]). Observable will take care of the user interface changes and does not bother about the changes or updated to the underlying or internal structure.

6.9.2 Menu Interface

Along with the text-area, menu interface is another crucial aspect of the PENG text editor user interface. It has been implemented using SuperFish. Using the updateAll() method, updates to the all the update table (lookahead info, and other outputs with which the user interacts) is taken care of.
This interface has different options. They are lookahead information, categories and words. In the lookahead information option, an indicator is used to let the user know that it is for lookahead information. The categories option lists all the available categories which appear to the user only when the user interacts with it by using the mouse hovering event or clicking it.

The words option or level has a list of all the words that corresponds to each category and appears only when the user interacts with it the same way as in the categories option (mouse hover or mouse click).
The coding to implement the menu itself was done in our HTML file. Following is the code for that :

```
1
2 <ul class="sf-menu" id="example" >
3 <li class="loadlook-div" class="current">
4
5 <a href="" data-bind="loadLookahead">Lookahead Information</a>
6 <ul data-bind="foreach: lookaheadObject">
7 <li class="current">
8 <a href="javascript:void(0);" data-bind="text:cat"> </a>
9 <ul data-bind="foreach: wfm">
10 <li>
11 <a href="javascript:void(0);"
12 data-bind="text:$data, click: $root.postWordClicked"> </
a>
13 </li>
14 </ul>
15 </li>
16 </ul>
17
18 </li>
19 </ul>
```

In line 8 cat refers to categories and in line 9 wfm refers to words from or words that are available. The above code outlines the thinking behind the way lookahead information works. The lookaheadObject has the data required for the lookahead information.
The lookaheadObject is a variable that consists of list of arrays. It is here where the words (wfm) are available which corresponding to the respective category (cat). Looking at line 6, using the foreach loop, for every category present in the lookaheadObject, there are words that correspond to the respective category.

6.10 Programming Methodology

6.10.1 Iterative prototyping

As with most software projects, this project followed iterative prototyping. Iterative prototyping is a design strategy where one works on a certain thing, checks it, if its not correct then go back to rectify it else move forward and do something new. The project will follow the plan as already discussed in section 4 (Timeline). Each week, during meetings with Dr Rolf
Schwitter, certain aspects and functionalities were discussed and agreed upon that needed to be added or changed in the project. Following is the workflow of the iterative process:

1. Talk about a functionality that needs to be added or changed and see if its feasible.
2. Work to implement the discussed functionality.
3. Checking if it works.
4. Repeat the process starting from step 1.

It was important to have iterative prototyping which made sure that weekly feedback was provided on the things that were being done. Feedback was provided by a crucial stakeholder of the project. It helped to detect error or whether the functionality is what it needed to be. This was necessary to make sure the project moved in the right direction. Once it was accepted then we would start the iterative process again. As talked about weekly feedback earlier, weekly meetings will be required for that. Attendance form for the weekly consultation meetings is available in Appendix

6.10.2 Basis of Programming Logic

The basis of code was Object-oriented programming (OOP). In OOP , the programming structure revolves around objects and the logic that acts upon those objects. In our project, objects comprise of attributes (data or adjectives) and methods (procedures or the verbs of the code).

OOP was used because it's a more flexible and intuitive approach towards coding and also it makes documentation and code maintainability easier. Debugging is also done comparatively easier using OOP. OOP helps to achieve more by writing less code which means the code size is reduced by a humongous level. Object-oriented.

Chapter 7

7. Working of Peng

The following sections will attempt to show how PENG^{ASP} works and try to clarify the working and how each part of the editor interacts and the logic behind it. To show the working of the PENG system, images have been used which show different stages of working. The images will also have captions or detailed explanation of what it is conveying to make it easier to understand.

Each image will be a different functionality. These functionalities have already been defined when requirements were being specified in the **section 5**. These functionalities include interacting with the text editor, typing in the text box, submitting text, saving a specification, loading a specification, etc.

7.1 Current State of Project

The figure below show the idle state of the PENG^{ASP} user interface. This figure will provide an overview and give clarity when the working of the PENG system is discussed.

The layout of the PENG^{ASP} (the system in idle state) is shown in the figure below. This layout has been divided into different parts. At the top there is a menu bar (needed for interaction) that the user can interact with in order to save or load specifications, and change the reasoner mode. At the right hand side it shows that the input mode is text mode, and it remains static as the speech input mode has been removed.

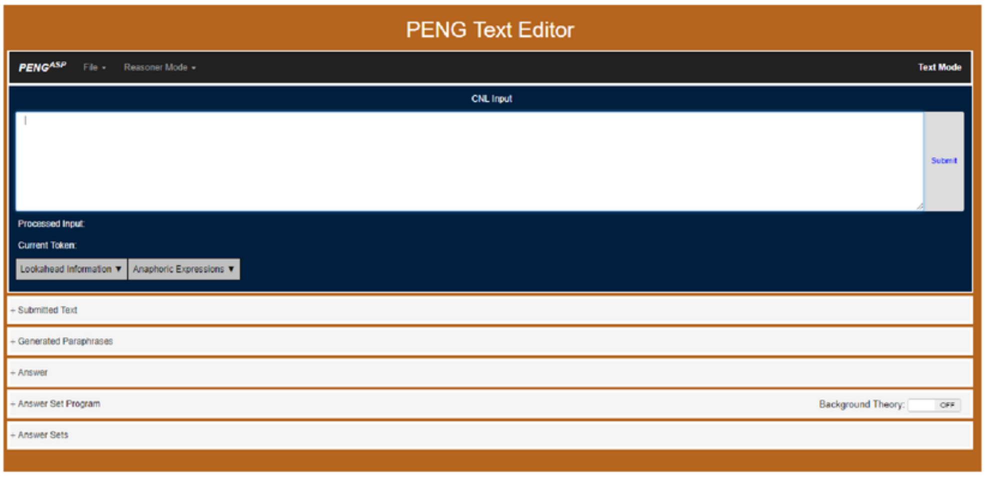


Figure 7 : Current User Interface

The text box at the centre is the area where the user types specifications to be executed. The text box is different from **figure** 4, as changes were asked to be made in the text box by an important stakeholder of the project. This change will be discussed at a later stage (**in section** 8) which talks about the changes made in this project which makes it different from the previous versions. In this text-box are the user can either input text or using lookahead information to populate the text box.

The last part is the div which shows the result of the submitted text. Following are the components of this result div :
Submitted Text : Shows the text that has been submitted either using the submit button or the enter key.
Generated Parapharased : Shows the paraphrases generated as a result of the entered text.
Answer Set Programs
Answer sets : Shows how the submitted text was sent to the lexicon.
Answer to queries.

7.2 Working Setup

To start using the interface and start the client-server communication is not that straightforward. Before we get started, we need to open SWI-Prolog and then compile the server. This is done by opening the prolog_server file in SWI-Prolog. In the previous version of the project (Bernardez's version), the PENG^{ASP} user interface would work properly only in the Google Chrome browser as the user interface would not render properly in Firefox or Internet Explorer. After the compilation of the prolog_server, we would open the PENG^{ASP} by typing http://localhost:8085/peng/ in the web browser.

HP Laptop 15-bs0xx was used for the project and the PENG^{ASP} was run on 4 web browsers. They are Google Chrome, Internet Explorer, Microsoft Edge, and Mozilla Firefox. The operating system used was Windows 10.

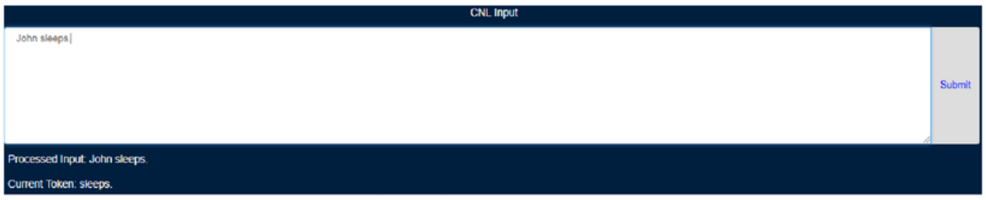
7.3 Demonstration of Interface Interaction

Menu-based interface and text-based interface will be discussed in the following section. This section will demonstrate the text-based and menu-based interface. Different text will be input for each of the interface.

7.3.1 Typed Input

In this section, working of the PENG^{ASP} while inputting the text manually (writing the specification) in the text area/editor is discussed. Using subsequent images, the way the system behaves and interacts will be discussed.

The figure below shows "John sleeps." being entered manually. When the user types this input, the token to be to be processed and sent to the server, is sent to the user using triggers. The triggers are a blank space, or a question mark, or a comma, or a full stop. Whenever one of the trigger is hit, the token is sent to the server. The triggers (or characters) are being detected and taken care by the keyEventHelper.js file as discussed in **section** 9 Before the user starts typing, it has to click inside the text area. As soon as the user clicks the same, a token is sent to the server which is a blank space (this happens every time, before a user starts typing) . When the user typed "John sleeps. "The tokens were sent as "John", then the blank space triggered the token, then "sleeps", and at last the full stop (another trigger). The blank space and the full stop acted as triggers in this specification. We would then get the lookahead data and anaphoric expression which happens because of the request sent to the PENG server (an AJAX response).


CNL Input

John sleeps

Processed Input: John sleeps.

Current Token: sleeps.

Figure 8 : Typing

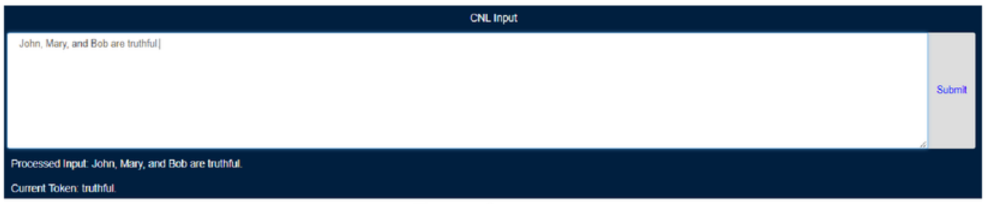
If we look at the prolog_server file, we got back the id of the the token and the following information:

```
Input: {
  "id":"2",
  "inputmode":"text",
  "editmode":"parse",
  "token":"John",
  "nbest":"[]",
  "featurestructure":"{ \"cat\" : \" \",  \"wfm\" : \" \"}",
  "filename":" ",
  "spectext":" ",
  "snum":"1",
  "spos":"0",
  "reasoner":"off",
  "reasonermode":"normal"
}

Input: {
  "id":"3",
  "inputmode":"text",
  "editmode":"parse",
  "token":"sleeps",
  "nbest":"[]",
  "featurestructure":"{ \"cat\" : \" \",  \"wfm\" : \" \"}",
  "filename":" ",
  "spectext":" ",
  "snum":"1",
  "spos":"1",
  "reasoner":"off",
  "reasonermode":"normal"
}

Input: {
  "id":"4",
  "inputmode":"text",
  "editmode":"parse",
  "token":".",
  "nbest":"[]",
  "featurestructure":"{ \"cat\" : \" \",  \"wfm\" : \" \"}",
  "filename":" ",
  "spectext":" ",
  "snum":"1",
  "spos":"2",
  "reasoner":"on",
  "reasonermode":"normal"
}
```

In the image below, we have another specification that has been typed John, Mary and Bob are truthful.



John, Mary, and Bob are truthful

Processed Input: John, Mary, and Bob are truthful.

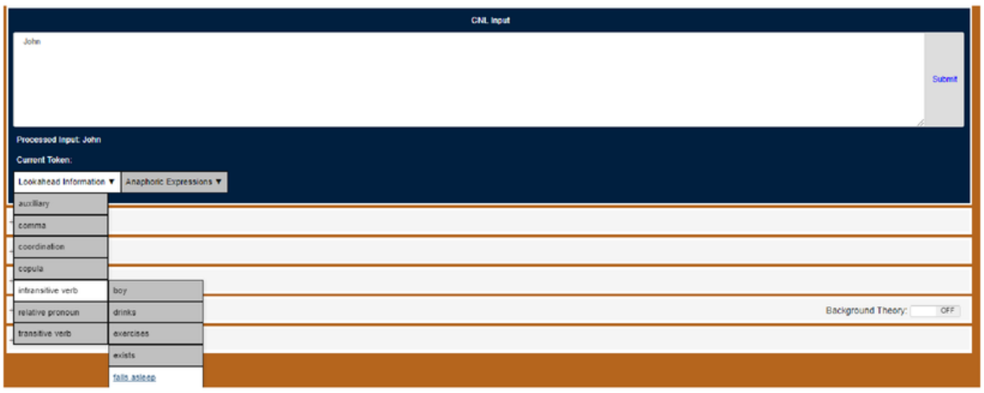
Current Token: truthful.

Figure 9

Here the tokens sent to the server are “John”, “Mary”, “ ,” , “and”, “Bob”, “are”, “runners”. As in the previous cases, the blank space again triggers the event to to let the system know that a token needs to be sent to the server. Then the server return the JSON object with a different value but the data remains the same. Each token is then sent one by one with the specific ID number that tells their position in the sentence. The tokens that are sent follow the same pattern as with the “John” token.
This to and from communication between the client-server goes on until the editor detect the full stop. When the full stop is detected, then the reasoner mode is turned on and the client-send communicates to the server and sends two JSON objects. These two JSON object are contains the values : “truthful” and the “.” token. “truthful” has the reasoner value: “off”, while the “.” token has the reasoner value switched to on. The system is coded in such a way that a full stop or a question mark marks the end of the sentence and lets it know that a sentence has been completed. Once the sentence has been completed and submitted, the server returns the following things as JSON object:
ID
Generated Paraphrases
ASP
Syntax tree
Anaphoric expression
Answer

7.3.2 Menu-based Interface

In this section, the menu-based interface is discussed, and a demonstration of using the menu-based interface to write specifications into the input area . How different things interact and behave in the client-server architecture is also discussed. Let’s start from the beginning when the user click in the input area, the same thing happens as when the user clicked in the input are while typing a specification, i.e. , a blank token is sent to the server which makes sure the server is listening and JSON object is created. Then the user types “John” and stops. Then it uses the lookahead information drop down menu to select the next word(s). From the drop-down menu, the user selects “falls asleep”. Here the trigger is the mouse click when the user clicks on the drop-down menu “lookahead information”. After the user selects “falls asleep” the lookahead information dropdown menu updates itself and predicts the next possible word. Here the user selects the full stop (.). This marks the end of the sentence as well. “The lookahead information” dropdown menu has different categories from which the user can select the next word. The user hovers over the category, the category gets highlighted and this opens another menu which consists of the word that could be selected. In our example, after the word “John”, the user then clicks on the lookahead information dropdown menu and chooses intransitive verb, which gives another list of words and chooses “falls asleep” from the given list. After this, the lookahead information gets updated so that the user can choose the next word. Then the user repeats the previous step, clicks on lookahead information, then hovers over full stop and then choose the full stop (.) punctuation mark. This, as in the previous cases marks the end of the sentence. Hence, the menu-based interface uses the same structure and logic when it comes to client-server communication, i.e., sending token from the client side to the server. Using the same code and logic, one can figure out where the errors are coming from and makes the system more efficient.



John

Processed Input: John

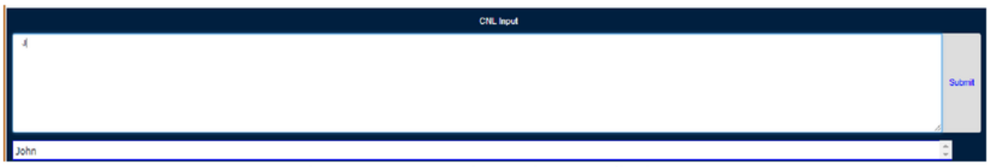
Current Token:

Background Theory: OFF

Figure 10: Screenshot of entering specification via menu interface

7.3.3 Using the prediction to enter specifications

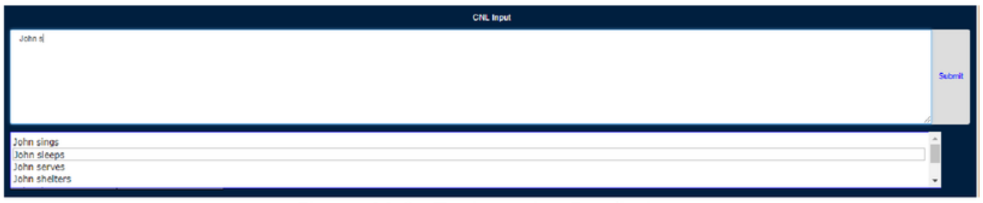
Another way to interact with the system and enter specification is to use the prediction box that appears when the user starts to type something. Following is an example :
The user clicks in the input area and it triggers the creation of the JSON object, in the same way as in the previous two sections. The user enters the character 'J' and there is a prediction of the word "John" that appears in the text area as seen in the figure below . After selecting "John" and hitting the spacebar, the blank space triggers the same events as before. Token ("John") is sent, and the lookahead information is also update. Then the user types the character 'S' and a bunch of predictions pop up. In this example, the user selects "John sings", and the input area gets populated with "John sings". This process can be repeated for different scenarios. But the sending of tokens, and trigger events remain the same as in previous cases (tokens get sent in the event of spacebar being hit, that is blank space in the input text area). **The following image** shows the step by step by selection of the sentence "John sleeps.



J

John

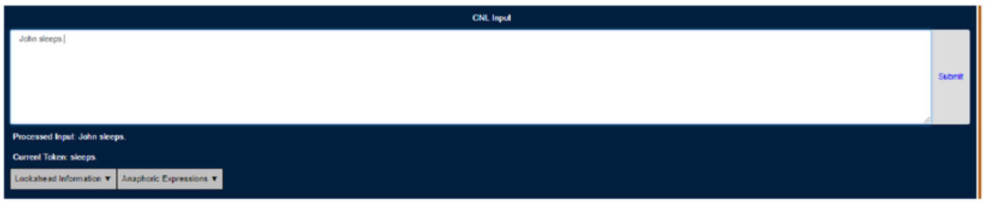
Figure 11.1 : Typing using Predictions



John s

John sings
John sleeps
John serves
John shelters

Figure 11.2 : Typing using Predictions



John sleeps

Processed Input: John sleeps.

Current Token: sleeps.

Figure 11.3 : Typing using Predictions

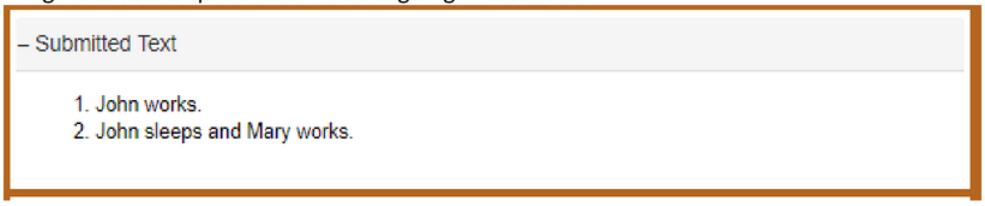
7.4 Saving, Loading, and Adding Demonstrations

This section will demonstrate the following user requirement functionalities as defined in **section**

- User Requirement 6: The user is able to load and save specifications.
- User Requirement 9: The user is able to add to the lexicon.

7.4.1 Saving Specifications

In this section the saving functionality of the system is shown and discussed. For this purpose, 2 sentences have been submitted and can be seen in the "Submitted Text". In other words, the processed specification will be the one that gets saved. The following image shows the specification that is going to be saved.



1. John works.
2. John sleeps and Mary works.

Figure 12 : Saved Specification in the Submitted Text box

Then the user hovers over to the **File** option in the menu bar and clicks it which triggers an event that shows a drop-down menu with the **Save** option as show in the following image

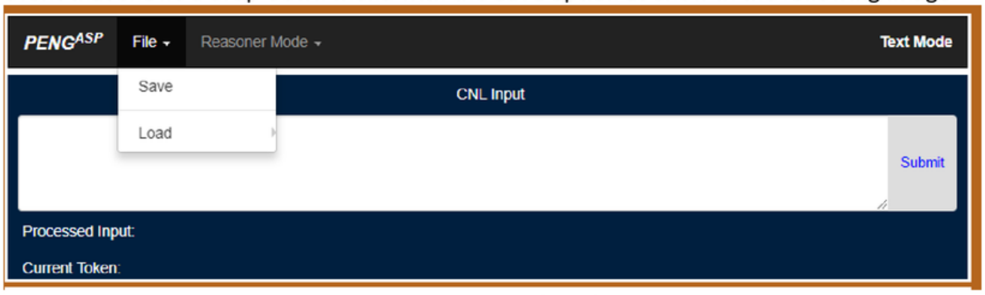
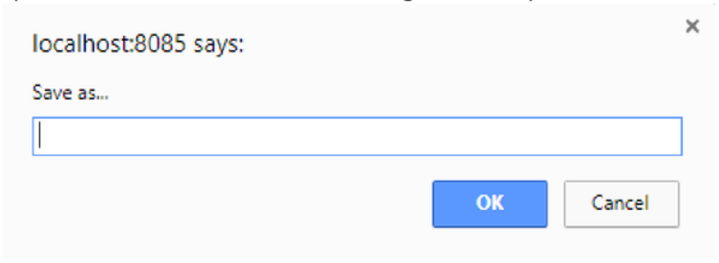


Figure 13 : Navigation to show how to save

When the user clicks on the save option, a dialog box opens which asks for the name of the specification to be saved as. The dialog box that opens can be seen below


localhost:8085 says:

Save as...

Then the user types in the required name and clicks on OK to save the specification. But, if the user thinks otherwise, and does not want to save the specification, it can click on Cancel and go back to the user interface. While saving the specification, if the user does not enter a name, an error message is seen. The error message reads " Please enter a file name".

The specification that is saved, is saved as a text file with the extension .txt . The file that has been saved can be seen when in the Load option inside the File option. In the next section, one of the examples shows a saved file that was loaded.

7.4.2 Loading Specifications

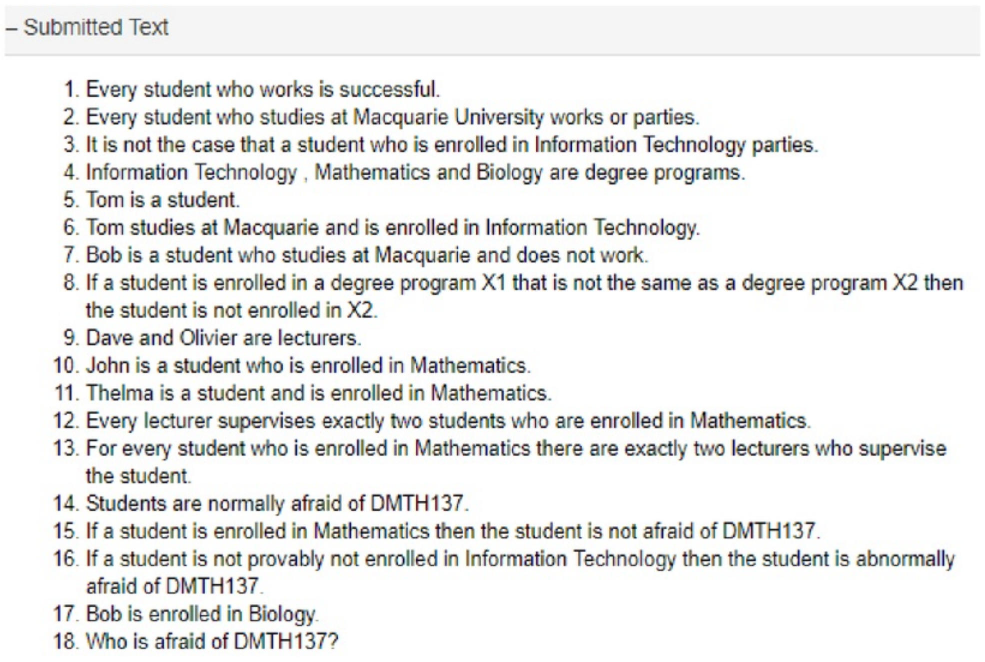
Similar to the previous section, this section will discuss and show the Load functionality of the PENG system. In order use this functionality, the user first navigate to the 'File' option in the menu bar and then click on it. This will show two options in the drop-down menu, Save and Load. Hover over to the Load option and click it. This again leads to a drop-down menu of already populated specifications saved as .txt files . Once a text file has been selected, the loading procedure begins. Like the save mode of the system, the load functionality needs to follow a certain pattern. A certain format has to be followed by the specification that are being loaded. Only the specifications that have already been processed, will be loaded. Which means the specification must have already been processed and submitted and seen in the Submitted text box. If this does not happen, then the user is provided with the following three options :

1. Save : This option arises in order to save the specification that has not been processed if the user wants to use it in the future.

2. Clear : This option will not save the specification but will load the specification. A user may want to use it in order to test the load functionality but does not want to save the specification.
3. Cancel : This is self-explanatory. If the user does not want to save the specification and load it then click cancel and return to the main user interface.

The following will show the loading functionality of the PENG system in three different ways, the first one, is to use the Load option from the File option and choosing an already saved specification. The second, is when we save a specification and load it. And the third, when we save a specification that has not been processed and load it.

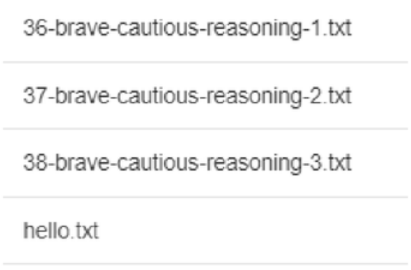
Below is the image of the first scenario. The specification can be seen loaded in the Submitted Text box. This is the file successful-student-owa-b.txt that is selected from the File->Load option.


– Submitted Text

1. Every student who works is successful.
2. Every student who studies at Macquarie University works or parties.
3. It is not the case that a student who is enrolled in Information Technology parties.
4. Information Technology , Mathematics and Biology are degree programs.
5. Tom is a student.
6. Tom studies at Macquarie and is enrolled in Information Technology.
7. Bob is a student who studies at Macquarie and does not work.
8. If a student is enrolled in a degree program X1 that is not the same as a degree program X2 then the student is not enrolled in X2.
9. Dave and Olivier are lecturers.
10. John is a student who is enrolled in Mathematics.
11. Thelma is a student and is enrolled in Mathematics.
12. Every lecturer supervises exactly two students who are enrolled in Mathematics.
13. For every student who is enrolled in Mathematics there are exactly two lecturers who supervise the student.
14. Students are normally afraid of DMTH137.
15. If a student is enrolled in Mathematics then the student is not afraid of DMTH137.
16. If a student is not provably not enrolled in Information Technology then the student is abnormally afraid of DMTH137.
17. Bob is enrolled in Biology.
18. Who is afraid of DMTH137?

Figure 14 : Specifications loaded and seen in the Submitted text box

Then comes in the second scenario where we first save the process input that was type by us. For this, the specification that was saved in the previous section will be used. The specifications have been saved as hello.txt as shown below

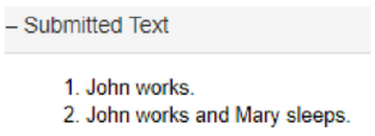

36-brave-cautious-reasoning-1.txt

37-brave-cautious-reasoning-2.txt

38-brave-cautious-reasoning-3.txt

hello.txt

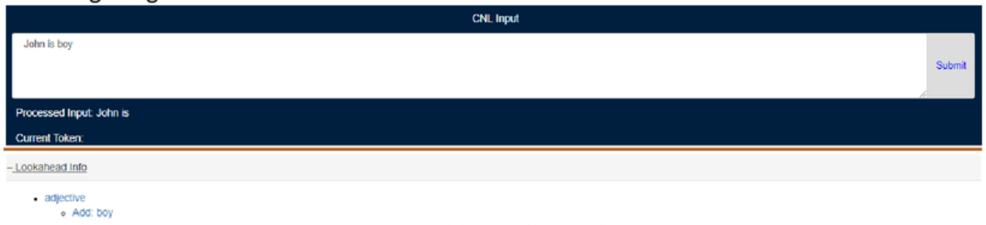
The following image shows the loaded specifications :


– Submitted Text

1. John works.
2. John works and Mary sleeps.

7.4.3 Adding Token to Lexicon

This section talks about another user requirement, which adding a token to the Lexicon. The adding functionality will also be demonstrated. When a user enters a word and that word is recognized as an unrecognized token, the system prompts and ask the user to add the token to the lexicon and make it recognizable. The following example will demonstrate this adding functionality:
The user types the following :
John is boy. Boy is not present in the lexicon as of yet. But this can be added as shown in the following image


CNL Input

John is boy

Submit

Processed Input: John is

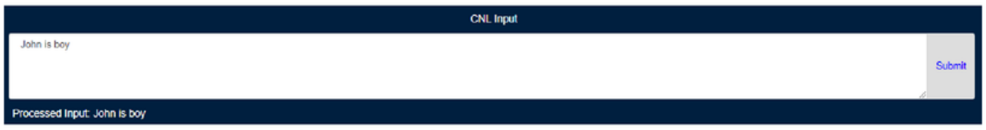
Current Token:

– Lookahead Info

- adjective
  - Add: boy

Figure 15: Adding token to lexicon

From the above image, we can see that the word boy is appearing in the text area, but this is not seen in the processed input which tells that the word is not in the lexicon and is unrecognized and if the user tries to type anything after boy, then an error message is shown. By clicking on Add:Boy , which can be seen in the Lookahead Info-> Adjective, the word boy is added to the lexicon and can been the seen in processed input as show by the image below:


CNL Input

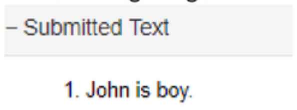
John is boy

Processed Input: John is boy

Figure 17 : Token added and processed

The word to be added will always be the first item in the appropriate category in the lookahead info dropdown menu.

The following image shows that John is boy has been submitted properly.



1. John is boy.

Chapter 8

8. Discussion and Current Work

8.1 Understanding the code and Documentation

The first part of the session has been dedicated to understanding the current system and how it works and doing necessary documentation for my own help and to assist those who work on this project in the future. The documentation will keep on improving when more coding will come into practice. As the project (client-side) is in HTML, CSS, and JavaScript, I had to take some classes online (at Udacity and an ongoing course at Udemy) on web development, and in particular, JavaScript.

8.2 User Interface design

As one of the areas to improve, user design needed some changes, as in my view, Bernardez's version was generic and seemed less appealing. Some of the code (Bernardez's code) has been changed to make the design look more appealing. Also, the entire view is centralized and more responsive.

Following is the screenshot after the change in user design:

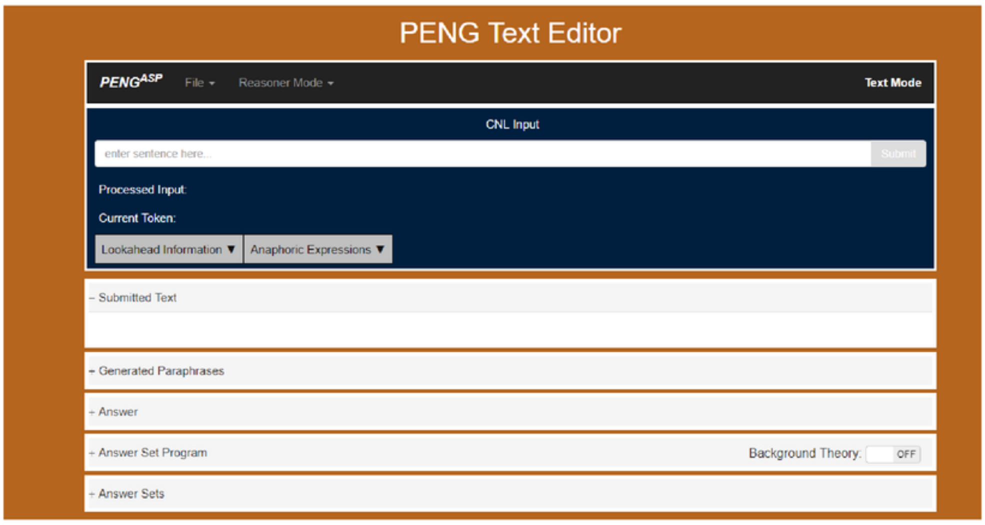


Figure 18: New User Interface (The text box shown in the image has been changed)

As seen from the screenshot, the layout is more fluid. Additionally, the layout is responsive. The input mode has been removed. Even though the speech input has been removed, it's API is still intact just in case we find a way to incorporate it into different web browsers. This layout is portable for all web browsers, which was not the case in Bernardez's version.

The following HTML code [14] was used:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">
```

The code infers the device, or the browser used and renders the HTML/CSS page in an optimized manner.

Apart from these changes, I had my own idea of how the user design could look at the end of this research project, one of the most important changes in this design is that the navigation bar remains fixed even when scrolled down which is more convenient to the user. The following is a hand-drawn diagram of the design:

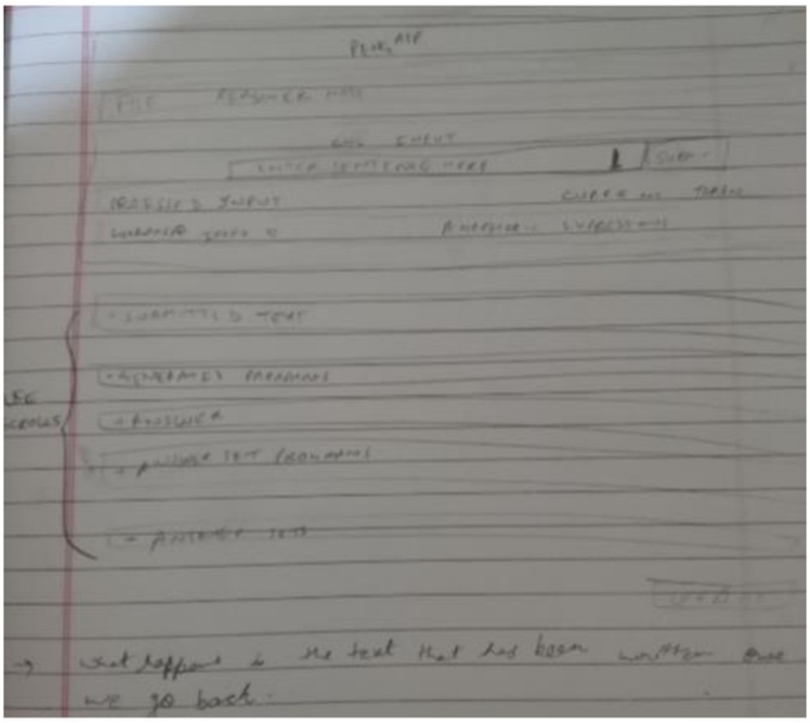

ENTER SENTENCE HERE
SUBMITTED TEXT
USE SCROLLS
what happens to the text that has been written once we go back.

Figure 19: What the user-interface could look like at the end.

In figure 8, the navigation bar is static so that the user does not have to scroll up and down again and again. Bottom right corner there is a Legacy button that will take the user to the previous version of the implementation.

Apart from this, I also made web landing page, which is responsive on all devices. It is a good idea to have a landing page, if and when, there comes a business side to this project. Following is the screenshot of the landing page:

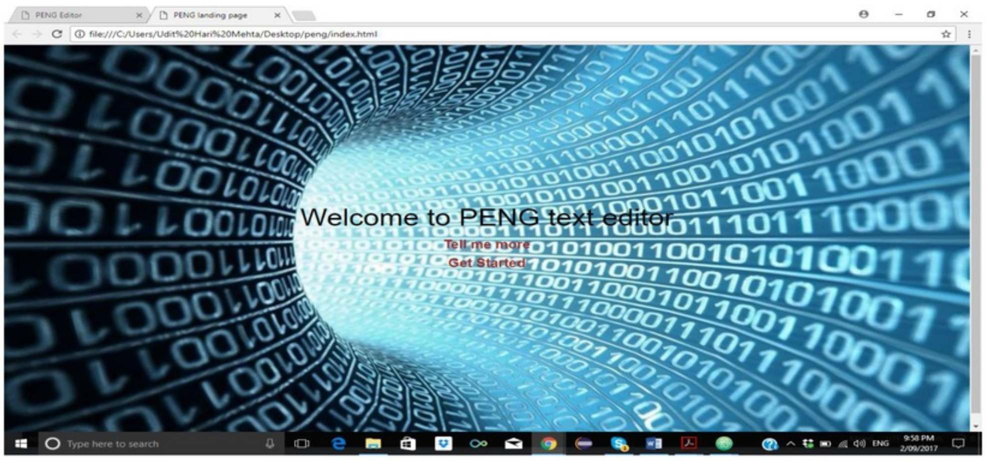

PENG Editor
PENG landing page
file:///C:/Users/Udit%20Hari%20Mehta/Desktop/peng/index.html
Welcome to PENG text editor
Tell me more
Get Started
Type here to search
9:58 PM
2/09/2017

Figure 20: Landing page

Tell me more link takes to a webpage which gives the description of the PENG system, in case the user wants to know more about the system. Whereas, the Get Started link takes the user to the localhost URL where it can interact with the system. Another screenshot to show the responsiveness is below:

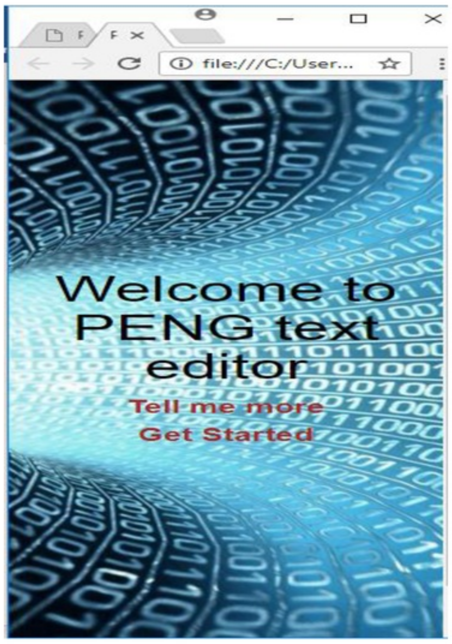


Figure 21: Landing page for mobiles

Following are the changes in the landing page, as the text was a bit difficult to read. The following CSS code was used to make the change:

background-blend-mode: screen; [15]

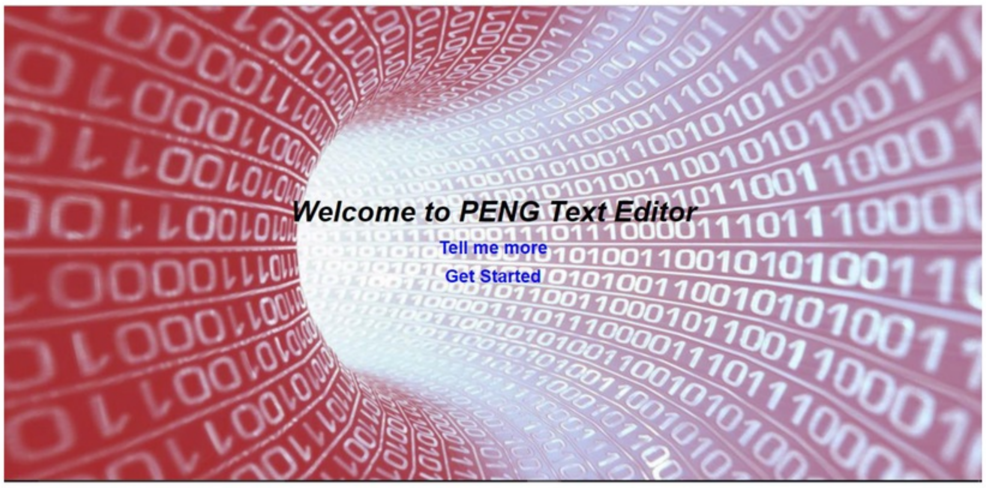


Figure 21: Updated Landing page

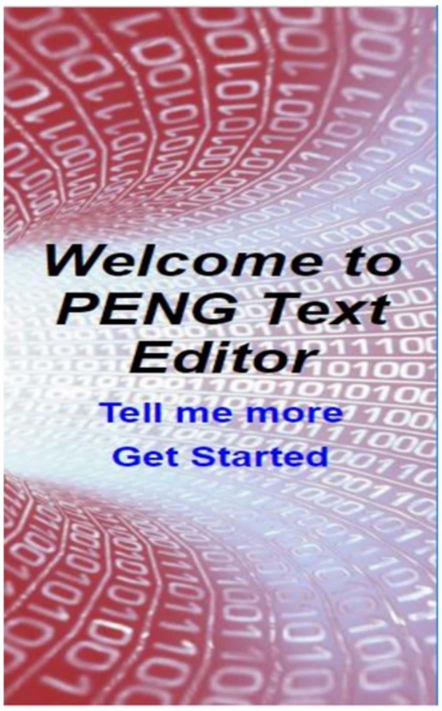


Figure 22: Updated Landing Page for Mobile

Another contribution is the making of a landing business page. Considering if the project goes live or onto the market, a business landing page is necessary to engage the customers.

From reading online and seeing videos about landing page, it is considered to be one of the most challenging things in front end development. As sometimes, the page is not rendered properly, or the background image is not flexible and does not occupy the entire screen all the time.

When we search something through search engines online, the first thing we see on clicking the result is a web page that is directed at the viewer so as to engage the user and make the product or business seem interesting and make the user to know more about the product and ideal cases, lead the user to buy the product.

CSS part

```
* {
  margin: 0px;
  padding: 0px;


}




body{


  margin:0;


  font-family:Arial,'Helvetica Neue', Helvetica, sans-serif;


  font-size: 17px;
  color: black;
  line-height: 1.6;
}


h1{
 color: black;
  font-weight: bold;
  font-style: italic;
}

a{
  font-size: 25px;
  font-weight: bold;
  color: blue;


}

#png{



  background-color:brown;
  background-image:
url('http://inside.mines.edu/UserFiles/Image/ComputerScience/CS%20web%20photos/cs%2
0research%20on%20big%20data.jpg');
  background-size:cover;
  background-position:center;
  background-repeat:no-repeat;
  height:100vh;
  background-blend-mode: screen;
  display:flex;
  color: black;
  flex-direction:column;
  justify-content:center;
  align-items:center;
```

```
}

#cnl{


color: #926239;

background-image: url('https://cdn.pixabay.com/photo/2017/01/06/19/15/soap-bubble-1958650_960_720.jpg');

font-weight: bold;

}
```

The most important part of the above code is manipulating the png id. The following code especially the height attribute of 100vh was the one that took the most time to figure out and done properly to make sure a landing page is made.

```
 background-size:cover;

 background-position:center;

 background-repeat:no-repeat;

 height:100vh;

 background-blend-mode: screen;
```

. . . .

HTML part

```
<!DOCTYPE html>

<html>

 <head>

  <meta charset="utf-8">

  <meta http-equiv="X-UA-Compatible" content="IE=edge">

  <meta name="viewport" content="width=device-width,intial-scale=1.0">

  <title>PENG landing page</title>

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta/css/bootstrap.min.css" integrity="sha384-/Y6pD6FV/Vv2HJnA6t+vslU6fwYXjCFtcEpHbNJ0lyAFsXTsjBbfaDjzALeQsN6M" crossorigin="anonymous">

  <link rel="stylesheet" href="main.css">
```

```
</head>


 <body>
  <header id="png">


    <h1 class="text-center"> Welcome to PENG Text Editor </h1>
    <a href="http://web.science.mq.edu.au/~peng/PengEditor.html" class ="button"> Tell
me more </a>
    <a href="#" class ="button"> Get Started </a>
  </header>


 </body>
 </html>
```

. . . .

The user interface of the has been changed with changes in the way it looks. Most importantly the input text box has been changed. In the previous versions, the <input> tag was used in the HTML file with attribute of type="text". The input box needed to be bigger and scrollable in case of larger specifications. This was not being achieved by just using the <input> tag, so it had to be replaced with <textarea>. Along with these certain changes were to be made in the CSS file to make sure the text box was of intended height and width.

The following HTML code helped achieve this :

```
<div class="input-group">


                    <textarea id="text_field" data-bind="event: {click: init, keypress: keyHandler,
keyup: backSpaceHandler }" spellcheck="false"  class="form-control awesomplete"
placeholder="enter sentence here..."> </textarea>
                    <span class="input-group-btn">
                         <button id="addButton-id" data-bind="event: {click: submitButton}"
class="btn btn-secondary" type="button">Submit</button>
```

```
                </span>
                </div>
```

. . . .

Changes in the CSS file

```
.searchbox-div {
    margin: auto;
    border: 3px solid #FFF;
    background: #001f3f; /*To change color of backgroun */
    padding: 10px;
    width: 95%;
    height: 210px;
    color: #FFF;


}


#text-field-div {
    width:100%;
    height:52%;
}
```

. . . .

Another addition to the PENG layout was to provide it with a header and title. To do so the following code was used :

In HTML file

```
<header  id="pg">
  <h1 class="text-center"> PENG Text Editor
  </h1>
```

In CSS file

```
#pg{
 color: white;
padding-top: 10px;

padding-bottom: 5px;

background-color: #B5651D;

}
```

. . . .

Another important change in the PENG system was the ability to write multiple sentences, as in the previous versions one could only type one sentence. This was hard to figure out as the existing code was not well documented and a lot of time had to be spent just to figure out what the code was doing and how it interacts with the rest of the files and how it interacts with the system. Also, not much was available on the Google Developers console to check what was going on. In the code that affected the ability to write multiple sentences, the code would not allow the user to type once the full stop had been entered, the key that would be hit after the first full stop would not be recognized and the user would not be able to see any input on the screen. Then, the change was made in the code where the key typed after the full stop would be recognized. This change was made in the KeyEventHelper.js file

Code before

```
  var keyID = e.keyCode == 13 ? e.keyCode : e.charCode;

        console.log(keyID)

        var keyVal = (String.fromCharCode(keyID));

        var charAllowed = (keyID != 13
&&viewModel.token().charAt(viewModel.token().length-1) != "." &&
viewModel.token().charAt(viewModel.token().length-1) != "?") || keyID == 13;
```

The above code would not accept any character after the full stop (.) or question mark (?) as it is not allowed by the var charAllowed.

. . . .

Code after

```
 var keyID = e.keyCode == 13 ? e.keyCode : e.charCode;
        console.log(keyID)
        var keyVal = (String.fromCharCode(keyID));
        var charAllowed = (keyID != 13 ) || keyID === 13;
```

After this another change was needed to be made, this change was in the punctuation function of the keyEventHelper.js file the following again shows code before and after

Code before

```
if (chr == '.' || chr == '?') {
            if(viewModel.allowInput && $.inArray(chr, viewModel.lookUpTable()) != -1)
                viewModel.postToken(chr);
```

. . . .

Code after

```
if (chr == '.' || chr == '?') {
            if($.inArray(chr, viewModel.lookUpTable()) != -1)
                viewModel.postToken(chr);
```

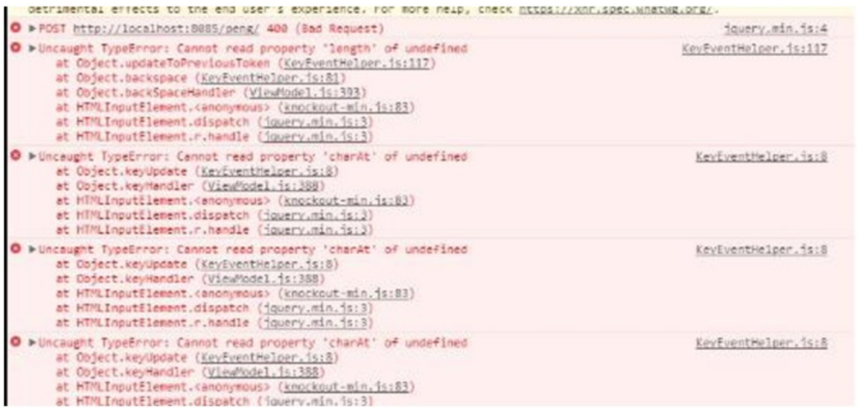
. . . .

Another change was to show the specifications of the file that was loaded onto the "Processed Input" on the user interface. This addition was made in the SuccessHelper.js file, it has a function loadSingleTextFile. To show the content or specifications of the load file , the following line was added

```
viewModel.textAreaStr(sentences);
```

. . . .

Another contribution to the project was the discovery of a bug. The system would crash when the backspace key was pressed multiple times successively when trying to delete a word or character from the specification. Following is the screenshot of the bug that was identified :



```
detrimental effects to the end user's experience. For more help, check https://xhr.spec.whatwg.org/.
✖ ▶POST http://localhost:8085/peng/ 400 (Bad Request)                                jquery.min.js:4
✖ ▶Uncaught TypeError: Cannot read property 'length' of undefined             KeyEventHelper.js:117
      at Object.updateToPreviousToken (KeyEventHelper.js:117)
      at Object.backspace (KeyEventHelper.js:81)
      at Object.backSpaceHandler (ViewModel.js:393)
      at HTMLInputElement.<anonymous> (knockout-min.js:83)
      at HTMLInputElement.dispatch (jquery.min.js:3)
      at HTMLInputElement.r.handle (jquery.min.js:3)
✖ ▶Uncaught TypeError: Cannot read property 'charAt' of undefined               KeyEventHelper.js:8
      at Object.keyUpdate (KeyEventHelper.js:8)
      at Object.keyHandler (ViewModel.js:388)
      at HTMLInputElement.<anonymous> (knockout-min.js:83)
      at HTMLInputElement.dispatch (jquery.min.js:3)
      at HTMLInputElement.r.handle (jquery.min.js:3)
✖ ▶Uncaught TypeError: Cannot read property 'charAt' of undefined               KeyEventHelper.js:8
      at Object.keyUpdate (KeyEventHelper.js:8)
      at Object.keyHandler (ViewModel.js:388)
      at HTMLInputElement.<anonymous> (knockout-min.js:83)
      at HTMLInputElement.dispatch (jquery.min.js:3)
      at HTMLInputElement.r.handle (jquery.min.js:3)
✖ ▶Uncaught TypeError: Cannot read property 'charAt' of undefined               KeyEventHelper.js:8
      at Object.keyUpdate (KeyEventHelper.js:8)
      at Object.keyHandler (ViewModel.js:388)
      at HTMLInputElement.<anonymous> (knockout-min.js:83)
      at HTMLInputElement.dispatch (jquery.min.js:3)
```

. . . .

As the project or the code was not documented at all and HTML, CSS, JavaScript, jQuery, AJAX had to be learnt as this was the first time I was introduced to front end development and a bit of back-end development, a lot of work had gone to understanding the entire system and different functionalities and interactions. Additionally, the code was documented nicely, which gives the reader a better understanding of the things involved in the system.

Chapter 9

9. Explanation of important JavaScript files

This section explains the main part of the project, which is the implementation of the JavaScript files and the logic behind it.

9.1 KeyEventHelper.js

The following code is from the **KeyEventHelper.js** file.

```
var KeyHandler = {

    keyUpdate: function(d, e) {

        var keyID = e.keyCode == 13 ? e.keyCode : e.charCode;

        console.log(keyID)

        var keyVal = (String.fromCharCode(keyID)); // Character form

        var charAllowed = (keyID != 13 ) || keyID === 13;

        //  var charAllowed = (keyID != 13 ) || keyID == 13;
```

The above code refers to the way text is input and allowed in the text area of the PENG text editor. The variable keyID checks what type of key has been pressed by the user as input. The keyCode of the key that has been pressed by the user can be seen in the console of the web browser using developer tools. Variable charAllowed allows other characters to be entered. There has been changes made to charAllowed which will be discussed in section (Solutions to limitations).

```
        if(keyVal == " " && viewModel.textAreaStr().length == 0)

            return false;

        if(!charAllowed)

            return false;

        if(viewModel.allowInput) {

            this.switchKeyVal(keyVal);

            keyVal = (keyID == 13) ? "" : keyVal;

            viewModel.textAreaStr(viewModel.textAreaStr()+keyVal);
            viewModel.updateLookUpTable();
        }
        else {
            var word = textLineData.nodes[textLineData.nodes.length-1];
            alert('"'+word+'" is not a valid token. Please go back and re-enter a token or add
to lexicon via predictive table');
            return false;
        }


        return true;
    },


    enterKey: function() {
     // changes made here


        var isEndOfSentence =
viewModel.textAreaStr().charAt(viewModel.textAreaStr().length === " ");
        // var isEndOfSentence =
//viewModel.textAreaStr().charAt(viewModel.textAreaStr().length-1) == "." ||
//viewModel.textAreaStr().charAt(viewModel.textAreaStr().length-1) == "?";
        if(isEndOfSentence) {
            var submitStr = viewModel.textAreaStr().replace(" .", ".");
            submitStr = submitStr.replace(" ?", "?");
            textLineData.addSentence(submitStr);
            viewModel.textList.push(submitStr);
            viewModel.textAreaStr('');
            viewModel.token('');
            viewModel.$text_field.val('');
            viewModel.init();
        }
    },
```

EnterKey function had been implemented to submit the entered text to the processor after which anophoric expressions and lookahead information could be seen. Additionally, this function helps for the text that was submitted to be seen in the submitted text div.

```
 punctuation: function(chr) { //str is what to update currentWord
        var sizeOfWord = viewModel.token().length;
        var prevWord = viewModel.token().slice(0, sizeOfWord);
        if(prevWord != "") {
            viewModel.postToken(viewModel.token().slice(0, sizeOfWord));
        }
        viewModel.firstIndexOfCurrentWord = viewModel.textAreaStr().length ;
        if (chr == '.' || chr == '?') {
            if( $.inArray(chr, viewModel.lookUpTable()) != -1)
                viewModel.postToken(chr);
            else {
                textLineData.sposNum--;
                viewModel.allowInput = false;
            }
        }
    },
```

The punctuation function was implemented to submit the text once a full stop (.) or a question mark (?) had been identified as the end of the sentence and there was nothing after the punctuation mark.

```
    backspace: function(d, e) {
        var keyVal = e.charCode;

        if (keyVal == 8) { //backspace detected
            if (viewModel.textAreaStr().length < viewModel.$text_field.val().length) {
                viewModel.textAreaStr(viewModel.$text_field.val()+" ");
            }
            var counter = 0;
            while (viewModel.textAreaStr() != viewModel.$text_field.val()) {
                viewModel.asyncFlag = true; // should be true
                viewModel.token(viewModel.token().slice(0, viewModel.token().length-1));
                var charBeingRemoved =
viewModel.textAreaStr().slice(viewModel.textAreaStr().length-1,
viewModel.textAreaStr().length);
                viewModel.textAreaStr(viewModel.textAreaStr().slice(0,
viewModel.textAreaStr().length-1));
                // When backspace all the way to previous token


                //var currentIndexInPreviousToken = viewModel.textAreaStr().length ==
viewModel.firstIndexOfCurrentWord-1;
                var currentIndexInPreviousToken = charBeingRemoved == " " ||
charBeingRemoved == "," || charBeingRemoved == "." || charBeingRemoved == "?";
                if (currentIndexInPreviousToken) {
                    var tokenToBeDel = textLineData.nodes[textLineData.nodes.length-2];
                    this.updateToPreviousToken(charBeingRemoved);


  if(tokenToBeDel != " ") {
                        var pop = textLineData.removeTailNode();
                        viewModel.postToken(pop);
                        viewModel.lookUpTable(viewModel.lookahead.wordTable);
                        viewModel.currentInitialLookUpTable =
viewModel.lookahead.wordTable;
                    }
                    else if(textLineData.nodes[textLineData.nodes.length-1] == " "){
                        viewModel.currentInitialLookUpTable =
viewModel.initSentenceLookUp;
                        viewModel.lookaheadObject(viewModel.initLookUpObj);
                    viewModel.lookUpTable(viewModel.initSentenceLookUp);
                }

                viewModel.allowInput = true;
            }
            else {
                viewModel.lookUpTable(viewModel.currentInitialLookUpTable);
            }
            counter++;

            if(counter > 50) {
                viewModel.textAreaStr(viewModel.$text_field.val());
                break;
            }
        }
        viewModel.allowInput = true;
        viewModel.asyncFlag = false;
    }
},
```

The above code is a to handle backspace key being pressed. It checks when the backspace is pressed and then checks the value in the text field that is being removed and then update the lookahead information when the backspace key is being pressed.

. . . .

```
updateToPreviousToken(charBeingRem) {
        var prevToken = textLineData.removeTailNode();
        if (prevToken == '.' || prevToken == '?')
            prevToken = textLineData.removeTailNode();
        viewModel.token(prevToken);
        viewModel.firstIndexOfCurrentWord = viewModel.firstIndexOfCurrentWord-
viewModel.token().length -1;

        if(charBeingRem == ",")

            viewModel.firstIndexOfCurrentWord++;

    },
```

The above function is needed to make sure the a character or word is deleted, the system knows that it has to make the previous word the current or the active word.

. . . .

```
    switchKeyVal: function(keyVal) {

        switch(keyVal) {

            case ',':

                KeyHandler.punctuation('');

                viewModel.token(",");

                break;

            case ' ': //SPACE CLICKED

                KeyHandler.punctuation('');

                viewModel.token('');

                break;

            case '.':

                KeyHandler.punctuation(keyVal);

                viewModel.token(viewModel.token()+keyVal);

                break;

            case '?':

                KeyHandler.punctuation(keyVal);

                viewModel.token(viewModel.token()+keyVal);

                break;

            case String.fromCharCode(13):  // Enter

                KeyHandler.enterKey();
                break;
            default:
                viewModel.token(viewModel.token()+keyVal);
        }
    },
```

The switchKeyVal function was required to know what would be done in case of the punctuation marks. Essentially, it is a punctuation handler and use the case statements (or switch case in Java) to implement the functionality. It looks at the punctuation mark, reads it and does the necessary work, i.e., that is either to post it as a token to the lexicon (in case of comma(,) or space(" ")) or to mark the end of the sentence (when the punctuation was either . or ? , provided there is a no other word after it).

. . . .

9.2 ViewModel.js

```
var viewModel = {
    $text_field: $("#text_field"),
    textAreaStr: ko.observable(""), // For dispaly
    result: ko.observable(""),
    firstIndexOfCurrentWord: 0,

    token: ko.observable(""),
    textList: ko.observableArray([]),
    smallAsp: "",
    bigAsp: "",
    aspState: false,

    $procSpan: $("#proc-span"),

    allowInit: true,
    // CLASSES
    lookahead: Lookahead,
    allowInput: true,

    catTable: ko.observableArray([]),
    anaExp: ko.observableArray([]),

    lookaheadObject: ko.observableArray([]),

    reasonerMode: "normal", // default settings
    inputMode: ko.observable("Text Mode"), //default settings
    speechData: ko.observable(""),

    $loader: $(".loader"),



    textParaList: ko.observableArray([]),
```

. . . .

The following function is implemented to submit the text and process it when the enter key is hit by the user.

```
    submitButton: function() {
        KeyHandler.enterKey();
    },
```

. . . .

```
    changeToTextMode: function() {
        $("#start_button").hide();
        this.inputMode("Text Mode");
        $('.searchbox-div').css('height', "220px");
    },
```

. . . .

As the speech mode was removed, the following code was removed as well.

```
changeToSpeechMode: function() {
    $("#start_button").show();
    $("#speech-detected").show();
    this.inputMode("Speech Mode");
    $('.searchbox-div').css('height', "240px");
    init();
    this.init();
},
```

. . . .

The following function changes the value of the reasoner mode to normal.

```
changeToNormal: function() {
    this.reasonerMode = "normal"
},
```

. . . .

The following function changes the value of the reasoner mode to brave.

```
changeToBrave: function() {
    this.reasonerMode = "brave";
},
```

. . . .

The following function changes the value of the reasoner mode to cautious.

```
changeToCautious: function() {
    this.reasonerMode = "cautious";
},
```

. . . .

The following function is needed to start the process of populating the lookahead information. this.init(); is only required at the start and then the lookahead table starts executing.

```
    loadLookahead: function() {
        this.init();
        var lookaheadTable = this.lookahead.createLookaheadTable(this.lookahead);
        this.lookaheadObject(lookaheadTable);
    },

    loadLookahead2: function() {
        var lookaheadTable = this.lookahead.createLookaheadTable(this.lookahead);
        this.lookaheadObject(lookaheadTable);
    },
```

. . . .

The following function takes data and event as parameters and posts the word as token by calling the postWordClicked function in the clickHelper.js file.

```
    postWordClicked: function(data, event) {
        clickHelper.postWordClicked(data, event);
    },
```

. . . .

The following function is similar to the the function above but in this case the anaphoric expression is checked as a valid token and submitted.

```
    postAnaExpClicked: function() {
        clickHelper.postAnaExpClicked(this);
    },
```

. . . .

More variables that are required to initialize the result text fields (Lookup, ASP, ANSWER)

```
    lookUpTable: ko.observableArray([]),
    asp: ko.observable(''),
    answer: ko.observable(''),
    currentInitialLookUpTable: [],
    initSentenceLookUp: [],


    asyncFlag: false,


    fileNames: ko.observableArray([]),
```

. . . .

The following function is required for the load functionality of the PENG system to work. Here the client interacts with the server side to fetch the file names that need to be loaded.

```
    loadFileNames: function() {


        if(viewModel.fileNames().length == 0) {
            var jsonObj = this.createJsonObject("load", " ", " ", " ", "off", "normal");


            $.ajax({
                url : "/peng/",
                type: "POST",
                data : jsonObj,
                success: function(data, textStatus, jqXHR)
                {
                    var json = JSON.parse(data);
                    fnames = json.filenames.slice(1,  json.filenames.length);
                    viewModel.fileNames(fnames);
                },
                error: function (jqXHR, textStatus, errorThrown)
                {
                    alert("Failed input on loading file names: \n "+errorThrown);
                }
            });
        }
    },
```

. . . .

```
    // Load a single file name
    loadFile: function() {

        var fileNameClicked = this;
        if (textLineData.nodes.length < 2) {
             //changes
            viewModel.loadFile1(this);
```

The following changes were made to attempt for the specifications in the loaded file to be displayed onto the processed input. But this would not work, but some changes were made elsewhere that helped achieve that.

```
          //  $("text_field").load(ViewModel.textList);
            //var randomString= textLineData.addSentence(this);
            //viewModel.textList.push(randomString);



        }
        else { // GOES HERE WHEN SPECIFICATION IS POPULATED


        $( "form" ).dialog({
            open: function() {
            // The submit button is hidden
            $( this ).find( "[type=submit]" ).hide();
            },
```

The following three option (Save, Clear and Cancel) is a part of the load functionality as discussed in **section**

```
buttons: [
    {
        text: "Save",
        click: function() {
            saveFile($("#textSave").val());
            textLineData.clearAll();
            viewModel.textList([]);
            viewModel.loadFile1(fileNameClicked);
            //var randomString= textLineData.addSentence(this);
            //viewModel.textList.push(randomString);
            $( this ).dialog( "close" );
        }
    },
  {
    text: "Clear",
    click: function() {
        textLineData.clearAll();
        viewModel.textList([]);
        $( this ).dialog( "close" );
    }
},
 {
    text: "Close",
    click: function() {
        $( this ).dialog( "close" );
    }
 }
]
});
```

```
        return false;

        }



        return true;

    },
```

. . . .

The following function is the one where the contents of the loaded file are processed and shown onto the submitted text. Here, the circular loader has been removed and replaced with an alert message. Again the communication takes place with the server and JSON object is sent. Then a call is made to loadSingleTextFile in the SuccessHelper.js file where the content of the loaded file is displayed.

```
loadFile1: function(fname) {


       //  viewModel.$loader.css("visibility", "visible");

       alert("Please wait while the specification is loaded");

        var jsonObj = viewModel.createJsonObject("load", " ", fname, " ", "off", "normal");

        var str = JSON.stringify(this);

        str = str.replace(" ", "");

        str = str.replace('"', "");

        str = str.replace('"', "");

        var index = viewModel.fileNames().indexOf(str);

        var fn = viewModel.fileNames();

        fn.splice(index, 1);

        viewModel.fileNames(fn);


        $.ajax({
```

```
        url : "/peng/",
        type: "POST",
        data : jsonObj,
        success: function(data, textStatus, jqXHR)
        {
            SuccessHelper.loadSingleTextFile(data, textStatus, jqXHR);
            //viemModel.textAreaStr(data);


        },
        error: function (jqXHR, textStatus, errorThrown)
        {
            alert("Failed JSON object input when loading file: \n "+errorThrown);
        }
    });



  // $.ajax({
      // url : "/peng/",
        //type: "GET",
        //data : jsonObj,
        //success: function(data, textStatus, jqXHR)
        //{

         //   $("#text_field").load(ViewModel.textList);


        //},
        //error: function (jqXHR, textStatus, errorThrown)
        //{
         //   alert("Failed JSON object input when loading file: \n "+errorThrown);
```

```
            //}
        //});



    },
```

. . . .

The following function is used to change the values of the lookahead information when a spacebar is clicked after a word has been typed.

```
    updateLookUpTable: function() {
        // This concat must be done to avoid unwanted changes in to other data structures
        var tempLookAheadTable = [].concat(this.lookUpTable());
        this.lookUpTable(this.lookahead.filterTable(this.token(), tempLookAheadTable));


    },
```

. . . .

The following function is needed to initialize the lookahead table when the specification is first loaded.

```
    init: function() {
        var lastNodePostedWasBlank = textLineData.nodes[textLineData.nodes.length-1] != "
";
        var textAreaEmpty = this.textAreaStr().length == 0
        if (textAreaEmpty && lastNodePostedWasBlank) {
            this.postToken(" ");
            this.initSentenceLookUp = this.lookUpTable();
            this.initLookUpObj = this.lookaheadObject();
        }
        else if (this.allowInput){
            this.$text_field.val(this.textAreaStr());
        }
    },
```

. . . .

```
    populateLookUpTable: function(data) {
        this.lookahead.setAll(data);
        this.lookUpTable(this.lookahead.getWordTable());
    },
```

. . . .

To populate the ANSWER div after specification has been submitted.

```
    setAnswer: function(ansData) {
        var ans = "";
        for(var s = 0; s < ansData.length; s++) {
            ans += ansData[s]+"\n";
        }
        this.answer(ans);
    },
```

. . . .

```
    setAsp: function(aspData) {
        var asp = aspData.asp;
        var clingo = aspData.reasoner;
        this.bigAsp = asp;
        var index = this.bigAsp.search("% -----------------------------");
        this.smallAsp = this.bigAsp.slice(0, index);
        if (this.aspState) {

            this.asp(this.bigAsp);

        }

        else {

            this.asp(this.smallAsp);

        }


        this.result(clingo);

        this.allowInput = true;

    },
```

. . . .

To take the word that has been input and process it as a token to be sent to the server using AJAX request.

```
    postToken2(word) {

        var wordData =  textLineData.createNode(word, this.reasonerMode);

        var request = $.ajax({

            url : "/peng/",

            type: "POST",

            data : wordData,

            async: this.asyncFlag

        });

        return request;

    },
```

. . . .

The following function takes target and arr as parameters and checks whether the target is part of the arr.

```
contains: function(target, arr) {
    for (i = 0; i < arr.length; i++) {
        if (target == arr[i]) {
            return true;
        }
    }
    return false;
},
```

. . . .

To keep on checking which key has been pressed and

```
keyHandler: function(d, e) {
    return KeyHandler.keyUpdate(d, e);
},


//fullStopHandler: function(keyVal)
//{
  //KeyHandler.punctuation(keyVal);
  //viewModel.token(viewModel.token()+keyVal);
//},
```

. . . .

To handle the backspace key so that that it can be used to delete words in the specification in the input area by going back in the specifications.

```
backSpaceHandler: function(d, e) {
    KeyHandler.backspace(d, e);
},
```

. . . .

```
createJsonObject: function(emode, token, fname, stext, reason, rmode) {
    var object = {
        id: -1,
        inputmode:"text",
        editmode:emode,
        token: token,
        nbest:"[]",
        featurestructure:"{ \"cat\" : \" \", \"wfm\" : \" \"}",
        filename: fname,
        spectext: stext,
        snum: -1,
        spos: -1,
        reasoner: reason,
        reasonermode: rmode
    }
    return object;
}



};

ko.applyBindings(viewModel);
```

. . . .

The following function is required for the save functionality of the PENG system which gives a prompt when the save option is clicked after clicking the File option.

```
function saveButton() {
    var file_name = prompt("Save as...");
    if (file_name != null) {
        if (file_name != "") {
            file_name += ".txt";
            alert(file_name);
            saveFile(file_name);
        }
        else {
            alert("Please enter a file name");
            saveButton();
        }
    }
};

var saveFile = function(file_name) {
    var spectext = "";
    for (var i = 0; i < textLineData.sentences.length; i++) {
        textLineData.sentences[i] = textLineData.sentences[i].split('\r').join('');
        spectext += ('\n'+textLineData.sentences[i]+'\n');
    }

    var saveData = {
        "id":"-1",
        "inputmode":"text",
        "editmode":"save",
        "token":" ",
        "nbest":"[]",
        "featurestructure":"{ \"cat\" : \" \",  \"wfm\" : \" \"}",
        "filename":file_name,
        "spectext":spectext,
        "snum":"0",
        "spos":"0",
        "reasoner":"off",
        "reasonermode":"normal"
    }
    $.ajax({
        url : "/peng/",
        type: "POST",
        data : saveData
    });
    alert("File saved");
}

var addToLexicon = function(cat, wfm, vform, num) {
    var idNum = textLineData.nodes.length ;
    var featstruct = FeatureStructure.createFS(cat, wfm, vform, num);
    console.log(featstruct);
    var addData = {
        "id": idNum,
        "inputmode":"text",
        "editmode":"add",
        "token": wfm,
        "nbest":"[]",
        "featurestructure":featstruct,
        "filename":" ",
        "spectext":" ",
        "snum":textLineData.sentences.length+1,
        "spos":textLineData.getSpos() -1,
        "reasoner":"off",
        "reasonermode":"normal"
    }
    $.ajax({
        url : "/peng/",
        type: "POST",
        data : addData
    });
}



$( document ).ready(function() {



$(".dropdown-menu").delegate("li", "click", function() {
    var str = JSON.stringify($(this).html());
    var addRegex = /Add/i
    var saveRegex = /Save/i
    var loadRegex = /Load/i

    var notFile = !(addRegex.test(str) || saveRegex.test(str) );

    if(notFile) {
        $(this).addClass("active").siblings().removeClass("active");
    }

});
```

End of ViewModel.js

. . . .

Chapter 10

10. Risk Assessment

The current system does not have a user login which can be a source of risk if some outside influence manages to corrupt what the user has been doing with the PENG editor.

The following are the other risks associated with the project:

10.1 Loss of data, Damaged Machine or Stolen Machine

As we know that a software project requires a lot of code, it is important to keep a copy of backup of the code/data that has gone into the project. In the event of loss of data that can happen due to a faulty machine or a stolen machine or in the event when the data is not saved, backup data can be a life saver.

Keeping regular backups and using GitHub to keep code are the ways to mitigate this risk.

At the moment, OneDrive is being used for backup. Once the documentation is complete, it will be pushed onto GitHub.

10.2 Time Management

Being a long project, managing time and completing activities as per the allocated time becomes necessary. Hence, time management is a risk where activities can get off track and project or the activities related to the project is not completed in due time.

This risk can be mitigated by keeping a good focus on the task in hand and following the Gantt chart meticulously.

10.3 Not Understanding Requirements

Going on with the project without understanding its requirement, or even worse, not knowing the requirements at all most likely leads to disaster(s).

But in this project, the requirements have been clearly explained by Dr. Rolf Schwitter, my supervisor, and thus the risk has been mitigated.

Chapter 11

11. Conclusion

In this paper, the PENG^{ASP} system's implementation and architecture has been kept forward. As discussed earlier, PENG^{ASP} is a system written in CNL. The text-based interface, menu-based interface, and the use of prediction text box has been discussed.

This paper presented the limitations of the existing PENG^{ASP} versions and tried to solve it. Changes in the user interface, code to overcome limitations, and documentation of the existing code formed the basis of the project. The speech interface has been removed as it was not deemed necessary considering how ineffective it was as the user had to speak the words loudly and slowly.

All the previous versions of the system allowed the user to type just the one sentence. This has been rectified and the user can now type multiple sentences.

To conclude, the use of multimodal interface is a good approach as it gives multiple options to the user to give the input and the makes the interaction easier and more enjoyable.

Chapter 12

12. Future work

The previous versions had the limitation where user can't type multiple sentences. This had been rectified, but there is still some bug with it. The sentences after the first full stop are not processed in an expected way, and some unintuitive things has to be done by the user to make sure that it is processed.

In the future, a speech interface or plug-in could be looked at the is cross-browser compatible and works smoothly and efficiently.

The bug that was discussed in chapter 8, where the system would crash after multiple backspace key press can also be fixed.

Chapter 13

13. Abbreviations

CNL	Controlled Natural Language
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
PENG	Processable English
ASP	Answer Set Programming
API	Application Programming Interface
ACE	Attempto Controlled English
URL	Uniform Resource Locator
CPL	Computer Processable Language
AJAX	Asynchronous JavaScript And XML
KR	Knowledge Representation

14. References

[1] https://en.wikipedia.org/wiki/Natural_language_processing

[2] R. Schwitter, "Controlled Natural Languages for Knowledge Representation".

[3] https://en.wikipedia.org/wiki/Basic_English

[4] R. Schwitter, "A Controlled Natural Language for the Semantic Web", Journal of Intelligent Systems, vol. 17, no. 1-3, 2008.

[5] https://en.wikipedia.org/wiki/Controlled_natural_language

[6] T. Kuhn, "A Survey and Classification of Controlled Natural Languages", Computational Linguistics, vol. 40, no. 1, pp. 121-170, 2014.

[7] T. Kuhn, "A Survey and Classification of Controlled Natural Languages", Computational Linguistics, vol. 40, no. 1, pp. 121-170, 2014.

[8] https://en.wikipedia.org/wiki/Definite_clause_grammar

[9] S. Guy and R. Schwitter, "The PENG ASP system: architecture, language and authoring tool", Language Resources and Evaluation, vol. 51, no. 1, pp. 67-92, 2016.

[10] cmusphinx/pocketsphinx, GitHub. [Online]. Available: https://github.com/cmusphinx/pocketsphinx.

[11] R. Schwitter, \Controlled natural languages for knowledge representation," Proceedings of the 23rd International Conference on Computational Linguistics: Posters, pp.1113-1121, 2010.

[12] Davis, R., Schrobe, H. and Szolovits, P. (1993). What is Knowledge Representation? AI Magazine, p.17.

[13] T. Stocky, A. Faaborg and H. Lieberman, "A commonsense approach to predictive text entry", Extended abstracts of the 2004 conference on Human factors and computing systems - CHI '04, 2004.

[14] "HTML meta tag", W3schools.com. Available: https://www.w3schools.com/TAgs/tag_meta.asp.

[15] “CSS background-blend-mode property", W3schools.com. Available: https://www.w3schools.com/cssref/pr_background-blend-mode.asp.

[16] D. Angluin, M. Frazier and L. Pitt, "Learning conjunctions of Horn clauses", Machine Learning, vol. 9, no. 2-3, pp. 147-164, 1992.

[17] N. E. Fuchs, K. Kaljurand, and T. Kuhn, \Attempto Controlled English for Knowl-edge Representation," Reasoning Web, vol. 5224, pp. 104{124, 2008.

[18] R. Schwitter, K. Kaljur, A. Cregan, C. Dolbear, and G. Hart, \A comparison of three controlled natural languages for OWL 1.1," Controlled Natural Language: Workshop on Controlled Natural Language, 2008.

[19] V. Lifschitz, \What is Answer Set Programming?" University of Texas, Austin, 2008, Accessed on: 08-08-2016. [Online]. Available: https://www.cs.utexas.edu/users/vl/papers/wiasp.pdf
[20] R. Schwitter, \Processing coordinated structures in PENG light," Advances in Arti-_cial Intelligence, pp. 658{667, 2011.

[21] P. Clark, P. Harrison, T. Jenkins, J. Thompson, and R.Wojcik, \Acquiring and Using World Knowledge Using a Restricted Subset of English," In FLAIRS Conference, pp. 506{511, 2005.

[22] The mvvm pattern," Microsoft Corporation
Available: https://msdn.microsoft.com/en-us/library/hh848246.aspx

[23] M. Wasson, \ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET," Microsoft, 2013 Available:
https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[24] Git." [Online]. Available: https://git-scm.com/downloads

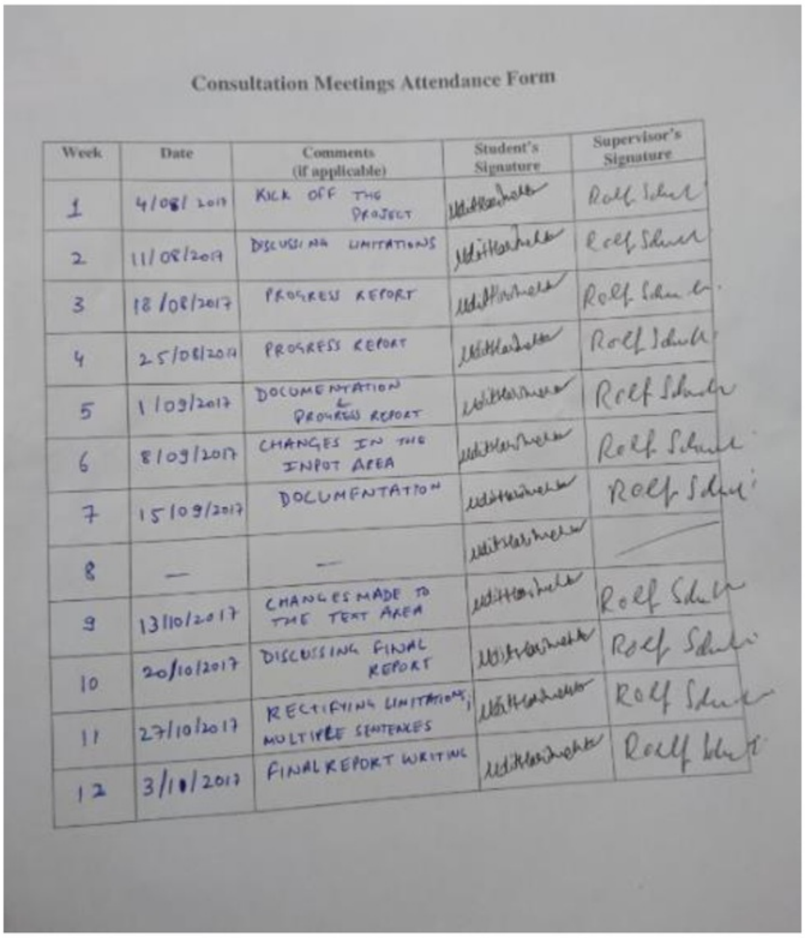
[25] Github Main Page," Github, Accessed on: 04-09-2016. [Online]. Available:
https://github.com/

[26] S. Sanderson, \KnockoutJS," Accessed on: 08-09-2016. [Online]. Available:
http://knockoutjs.com/

[27] J. Resig, \JQuery," Accessed on: 04-09-2016. [Online]. Available: https://jquery.com/

Appendix A

Meeting Consultation Form



Consultation Meetings Attendance Form

Week	Date	Comments (if applicable)	Student's Signature	Supervisor's Signature
1	4/08/2017	KICK OFF THE PROJECT	[illegible]	[illegible]
2	11/08/2017	DISCUSSING LIMITATIONS	[illegible]	[illegible]
3	18/08/2017	PROGRESS REPORT	[illegible]	[illegible]
4	25/08/2017	PROGRESS REPORT	[illegible]	[illegible]
5	1/09/2017	DOCUMENTATION & PROGRESS REPORT	[illegible]	[illegible]
6	8/09/2017	CHANGES IN THE INPUT AREA	[illegible]	[illegible]
7	15/09/2017	DOCUMENTATION	[illegible]	[illegible]
8	—	—	[illegible]	
9	13/10/2017	CHANGES MADE TO THE TEXT AREA	[illegible]	[illegible]
10	20/10/2017	DISCUSSING FINAL REPORT	[illegible]	[illegible]
11	27/10/2017	RECTIFYING LIMITATIONS; MULTIPLE SENTENCES	[illegible]	[illegible]
12	3/11/2017	FINAL REPORT WRITING	[illegible]	[illegible]

Appendix B

Tool List

1. OneDrive
2. DropBox
3. SWI-Prolog
4. Git and GitHub
5. JS
6. HTML
7. CSS
8. KnockoutJS
9. jQuery
10. HP Laptop 15-bs0xx
11. Internet Explorer
12. Mozilla Firefox
13. Google Chrome
14. Udemy's Web Developer BootCamp
15. ViewModel.js written by Evan Bernardez and modified by me