A MULTIMODAL INTERFACE FOR CONTROLLED NATURAL LANGUAGE PROCESSING

Rudz Evan Bernardez

Bachelor of Engineering
Software Engineering Major

Department of Electronic Engineering
Macquarie University

November 7, 2016

Supervisor: Dr. Rolf Schwitter

ACKNOWLEDGMENTS

I would like to first express my deepest appreciation to Dr. Rolf Schwitter for providing me with his guidance, support, and knowledge throughout this whole project. His active role from beginning to end has played a large role in me completing this project, without his guidance, I would not have had the same learning experience.

I would also like to thank my family and friends who have supported and encouraged me throughout my whole undergraduate program.

STATEMENT OF CANDIDATE

I, Rudz Evan Bernardez, 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: Rudz Evan Bernardez

Student's Signature: Rudz Evan Bernardez (electronic)

Date: November 7, 2016

ABSTRACT

Over the last 15 years, speech recognition improved dramatically and sophisticated technology has become more available. It has encouraged a project that explores the integration of a speech interface in a controlled natural language (CNL) system, which prior to this, had yet to be explored. A CNL is a subset of a particular written natural language (base language) which has been engineered to restrict language components such as syntax, lexicon, and/or semantics. By using a speech interface API, this paper presents an implementation of a predictive editor that integrates text and speech as inputs in a CNL processing system, namely, PENG^{ASP} Multimodal. In addition to building the system, a study was conducted to determine the practicality a speech interface is in a CNL system. In our findings, using speech input demonstrated to have significant benefits when looking at the speed during user input; however, the speed significantly declined as the number of errors during speech recognition grew.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, natural language processing has shown great advances in analysing, understanding, and generating languages that appear natural. Those advancements have largely been due to the recent progresses made in computing power, the availability of large scale annotated corpora, machine learning, and the understanding of structures of human language and its practical usage [1]. This led to the birth of widely used intelligent personal assistant programs such as SIRI [2], Google Now [3], and Cortana [4]. However, these programs and other natural language processors are still limited and basically rely on sophisticated forms of pattern matching.

Many literatures [1, 5, 6] credit the highly expressive nature of natural language to be the root cause of its complexities, while for humans, this expressive quality is what makes natural language a powerful tool for communication. An extreme approach to meet the challenges that come along with natural languages is to use or create a heavily constrained language such as those found in formal languages. This can prevent the language from containing unambiguous syntax and semantics, which then can be used as knowledge representation languages. However, this heavily restrictive form will diminish the natural properties required in order for a language to be practical and understandable. An approach that allows us to preserve a respectable level of a language's natural properties while reducing (or potentially eliminating) ambiguity is to use a controlled natural language (CNL). A CNL can be engineered to meet the required level of naturalness for human understanding and to be restricted enough to enable automated reasoning [6].

An issue with CNL based systems is the learning stage required in order to use them effectively. Each CNL has various restrictions, as such, we cannot expect an author to learn and remember each restriction in the language. In order to facilitate this issue, a popular approach used by various CNL systems such as PENG^{ASP} [7] and AceWiki [8] was a predictive editor. A predictive editor is an authoring tool that assists authors to creating content that is syntactically correct in a word by word basis. This approach can enforce the language restrictions and guide authors without them requiring any training in the language. A predictive editor can come in the form of a menu-based interface; for this, the user selects words generated from predefined vocabulary [9].

The predictive editor and menu-based interface are still lacking in regards to naturalness. An approach used by smart agents with natural language processing technology is the use of a speech recognition interface in addition to the standard text input. The obvious reason to create such a system is to provide a more natural conversation for when machine and a user (human) communicate [1]. This idea of a multimodal interface has yet to be explored in CNL based systems.

1.1 Project Goal

The goal of this project is to create a multimodal interface for the computer-processable CNL called PENG (Processable English). This interface will be a web-application that will be used as an authoring tool in the PENG CNL without requiring users to learn the controlled language. This multimodal aspect of the application will consist of three methods for user input: (i) typing into a text field, (ii) selecting items from a lookahead menu table, and (iii) using speech as input.

It is important to note that a prototype of various components of this application has been built previously by a student; however, this implementation will be rebuilding it from the ground up. The reason for the reconstruction was largely due to the various bugs and performance issues found. Furthermore, the exisiting prototype's implementation was extremely hard to maintain and rebuilding from the ground up would have required less effort. In addition to improving the system, the project also aims to determine the practicality of a speech interface in a CNL system.

1.2 Project Planning

This section outlines the scope, activity breakdown, and cost of the project.

1.2.1 Scope

The project focuses on a machine-oriented English-based CNL by exploring a multimodal interface for controlled natural language processing. More specifically, we looked at how speech and text can be integrated with an existing CNL processor. By allowing such inputs, a more effective study can be conducted on determining the practicality a speech interface was in a controlled natural language system.

In addition, two other studies was conducted; the first compares the new system with the existing, and the second compares two speech interfaces, that is, Google Web Speech API [30] and IBM Watson Speech-to-Text [31].

1.2.2 Activity Breakdown

The project will be divided into six tasks and each task will have an allocated start time and an expected end time as shown in table 1.1. The following tasks are:

1. Research on various technologies to be used for implementation.
2. Implementing the predictive editor.
3. Refining the interface for the predictive text editor.
4. Integrating the speech interface.
5. Refining code; this includes any tasks to improve code organisation, performance, and maintainability.
6. Report writing

	25/07/2016 – 07/11/2016														
Weeks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Task 1	▨	▨													
Task 2			▨	▨	▨	▨	▨	▨							
Task 3					▨	▨	▨	▨	▨	▨	▨				
Task 4							▨	▨	▨	▨	▨	▨			
Task 5													▨		
Task 6														▨	▨

Table 1.1: Timeline for each task

1.2.3 Project Cost

The project was assigned a budget of $300 Australian dollars. However, the required resources were either already in my possession or free software which can be downloaded online. Therefore, the project does not require any financial expenses.

Chapter 2

Background and Related Work

2.1 Controlled Natural Languages

A Controlled Natural Language (CNL) is a subset of a particular written natural language (base language) which has been engineered to restrict language components such as syntax, lexicon, and/or semantics. It preserves majority of the natural properties of its base language so speakers of its base language can fully understand the CNL. [6, 10].

CNLs can be generally grouped into two categories; the first is human-oriented, the second is machine-oriented. These two categories are based generally on what the CNL is designed to accomplish [6].

2.1.1 Human-oriented CNLs

Human-oriented CNLs are designed to improve readability and understandability in technical documents, particularly for non-native speakers. The earliest CNL documented, Basic English [11], was first presented in 1930 and is considered the most influential human-oriented CNL. This led to a birth of human-oriented controlled languages with no particular goal in mind with regards to domain and community [6]. Until years laters, an influential CNL, namely Simplified Technical English (STE) [12], was developed for the aerospace industry.

STE was created with the intention of it being used for maintenance manuals, this later expanded to technical publication specifications and official directives. It claims to improve understandability, translatability, reliability, and comprehension while reducing ambiguity. STE has been the standard for much of the technical documentation in the aerospace industry [12].

Developments of human-oriented controlled languages began during to what Kuhn [10] refers to as the general era and the technical era. These two eras are part of the other eras Kuhn has observed with overall themes when discussing the evolution of CNLs. The general era ended around the late 1960s or early 1970s, and were designed to improve human comprehension. The next era that followed was the technical era, this began in the early 1970s, much of the CNLs in this era were used in technical documents to improve human comprehension and machine translation. Finally, beginning in the mid 1990s, the logical era designed CNLs to allow for some form of mapping to formal logic. However, even in the most recent era, the logical era, human-oriented controlled languages still appear in various work. For example, Simplified Wikipedia [13], a version of the online encyclopaedia Wikipedia, uses Basic English and Special English to form its articles; both of which are controlled languages developed from the earlier eras. In contrast to Basic English, the articles in Simplified Wikipedia are written to improve understandability through using simpler word structure and syntax, a common goal amongst human-oriented CNLs.

2.1.2 Machine-oriented CNLs

Machine-oriented CNLs are designed to improve the translatability of technical documents and for writing text that can easily be translated into a formal language [6]. CNLs such as Caterpillar Technical English (REF), Perkins Approved Clear English (PACE) (REF), and various machine-oriented controlled languages started to appear as early as the 1970s. Earlier versions of machine-oriented languages were used more for the translatability and simplification of technical documents [10].

2.1.3 Examples of Machine-oriented CNLs

In the last two decade, CNLs have focused more on being machine-oriented and to allow for mapping between CNL and formal logic. There are three influential general-purpose controlled languages Schwitter [6] compares: Attempto Controlled English (ACE), Processable English (PENG), and Computer Processable Language (CPL). We refer to them as general-purpose in a sense that they were created with no specific application domain or situation in mind. These languages share two properties:

1. The text is constructed in a way that writing and comprehending becomes easier.
2. They are defined in such as way that they can be automatically translated into a formal target language and used for automated reasoning.

2.1.3.1 Attempto Controlled English (ACE)

ACE was designed to provide domain specialists with an expressive knowledge representation in unambiguous English that is easy to write and understand (REF. ACE works by automatically translating ACE text unambiguously into first-order logic using Attempto Parsing Engine (APE). The language supports complex noun phrases, plurals, anaphoric references, subordinated clauses, modality, and questions. ACE texts itself appear quite natural and can easily be understood by readers like all controlled languages developed specifically for human comprehension. However, unlike those languages, the learning required to write in ACE text is supported with an editor; thus, allowing users to construct correct ACE texts [6, 14, 15].

2.1.3.2 PENG^{ASP}

The PENG^{ASP} system [7] is an authoring tool for writing non-monotonic specifications and translates the specifications through discourse representation structures into a modelling language that allows for solving complex problems. This simple and powerful modelling language is called Answer Set Programs (ASP), it is a form of declarative programming and is particularly useful in knowledge-intensive applications [16]. The language processor of PENG is based on a grammar that is based on a definitive clause grammar (DCG) style notation and processed by a chart parser. This is different to ACE as PENG^{ASP} uses ASP as target language [17].

The PENG system, like ACE, does not require writers to remember the structure of its controlled language. The systems that have been built with the PENG CNL parse specifications at a token level; as such, it has allowed for predictive editing functionalities. These predictive functionalities have been used to guide users by dynamically enforcing the grammatical restrictions of the language [6,15]. More of existing PENG based systems will be discussed at Section 2.2.

2.1.3.3 Computer Processable Language (CPL)

CPL was developed at Boeing Research and Technology and translates English sentences to a formal frame based knowledge representation (KR) language [18]. The CPL interpreter works out various types of ambiguity by depending on heuristic rules for language processing activities. In contrast to ACE, which utilises a small set of strict interpretation rules, and PENG, which is supported by a predictive text editor. Texts come in three types: ground facts, questions, and rules. Sentence types have the following form: grounds facts takes three forms [6]:

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

Questions come in five main forms, the main two are:

"What is NP?"
"Is it true that sentence?"

For rules, CPL accepts a pattern in the form of:

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

2.1.4 Evaluation of ACE, PENG^{ASP} and CPL

The ACE editor simplifies using ACE by providing writers with a text editor that can be used to help construct ACE texts; PENG uses a predictive editor to assist users in constructing PENG texts; and CPL works out various types of ambiguites during its language processing activities. All three languages also have natural appearances making comprehending easier. Although the three languages mentioned have tools to support writers, they still require some skills in the controlled language to be used competently as these editors only guarantee syntactically correct sentences be constructed. However, it still remains that writers benefit using these languages as it minimises learning efforts, as oppose to using a language such as Basic English, which provides no support tools for writing, largely due to not being able to correspond to a formal language. As for property 2, mentioned at Section 2.1.3, where the language is defined in such a way that machine translation is possible; ACE, PENG, and CPL satisfies it. Property 2 is satisfied as: ACE translate ACE text to first-order logic; PENG translates PENG text to ASP; and CPL can translate English sentences to formal knowledge representation (KR).

2.2 Existing PENG^{ASP} Projects

As mentioned in Section 1.1, this project is a reconstruction of an existing prototype; the reason for the reconstruction is due to the various issues relating to stability and missing functionalities identified in the exisiting prototype. In this implementation, we hope to resolve those issues and create a more stable application.

The existing work implemented by Nalbandian [19] is an extension of the PENG ASP web-based predictive text editor implemented by Guy [7]. Guy's implementation is a web application that has an input area to allow for user input and a menu-based interface to guide users in writing specifications, as shown in Figure 2.1. Nalbandian and Scwitter's implementation is an extension of Guy's implementation, with the addition of a speech interface, as shown in Figure 2.2.

For both implementations, a client-server architecture was used with the client-side implemented in HTML, CSS, and Javascript. As for the server-side, Prolog was used to implement it due to its roots in first-order logic. In addition to those languages, libraries such as SuperFish (for the menu-based interface) and JQuery were used.

The changes that will be done in this project are the entire code base in the client-side; this is where majority of bugs can be traced from. The existing prototype lacked the use of programming language libraries that simplified tasks such as data-binding, which was not considered at all. It also lacked proper code organisation that made it unmaintainable and unnecessarily large in code size. Overall, we wanted to preserve the core functionalities and the general layout of the existing system's interface.

2.3 Multimodality for CNL Interfaces

One of the limitations found using speech inputs was the inconsistency in results from the speech recogniser. This was typically due to slight mispronunciations and background noises, which in most cases, will not be misinterpreted by an English speaking person. For that reason, it was necessary to integrate it with a text and a menu-based interface to support and improve usability.

The use of a text-based interface has been a popular choice amongst machine-oriented CNL systems such as PENG [7] and ACE [8]. These systems have typically used two methods when parsing input. The first is to allow the user to enter a sentence or sentences and then communicate with the CNL processor. The second does not wait for the user to complete a full sentence and instead communicates to the CNL processor whenever a new word is entered. An advantage for the first method is that it avoids the constant communication with the language processor required in second method. The constant communication can add complexities to a CNL based system in terms of performance and implementation; however, the additional functionalities that can be introduced to the CNL system when using the second method will significantly improve usability. The second method also conforms well with how the PENG server requests inputs, which operates similar to a stack. Therefore, choosing the second method for parsing was the approach taken for this project.

The menu-based interface is an approach that has been previously explored on lan-

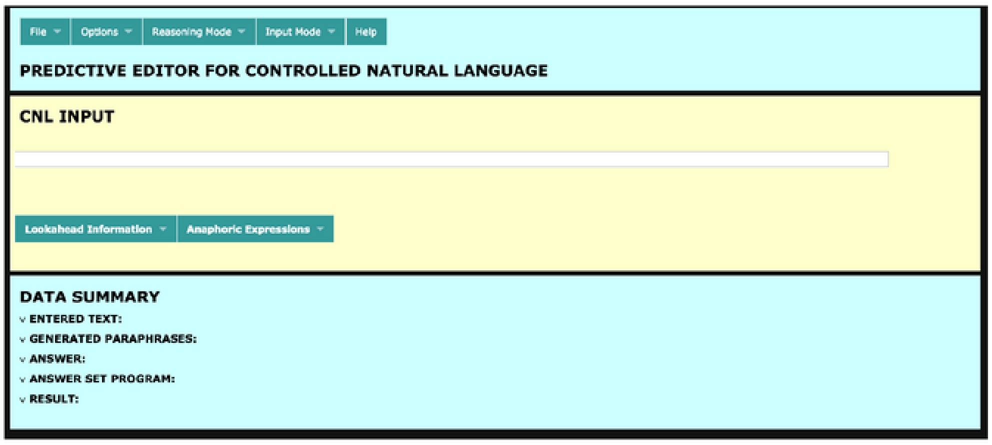


Figure 2.1: Screenshot of Guy's PENG web-based text editor application.

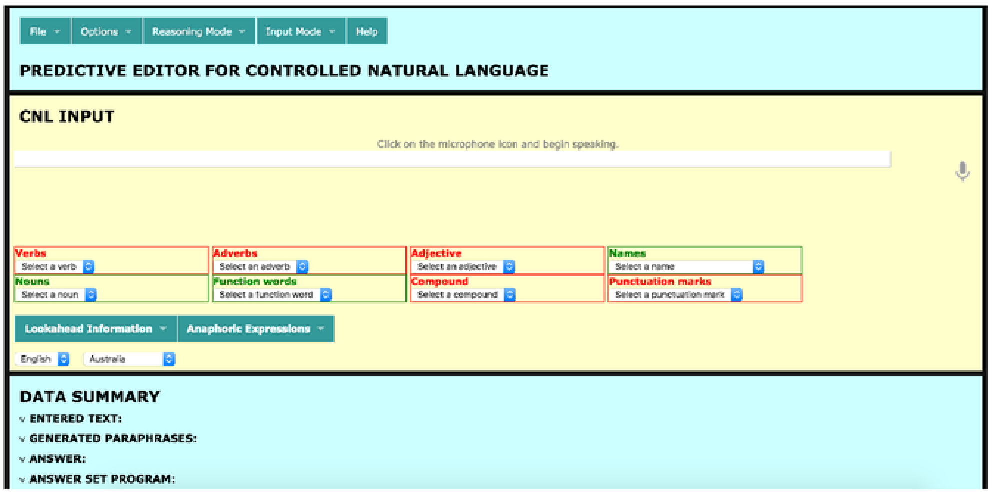


Figure 2.2: Screenshot of Nalbandian's PENG web-based text editor with speech application.

guage processing systems. One in particular, was the natural language system named NLMenu [20]. NLMenu was one of the earliest language processing systems with a menu-based interface to be implemented. It worked by dynamically displaying available words through its predictive parser that retrieves data from a semantic grammar and lexicon.

Years later, language processing systems have continued to use the same or similar approach when integrating a menu-based interface into their own systems [7,8,19]. By using this approach when building a menu-based interface, it guarantees that the language processor is able to understand all inputs thus assuring complete correctness in user inputs when parsing. Therefore, it is only natural that NLMenu's approach should be used as the basis for this project's menu-based interface.

Speech interfaces in language processors has grown recently in popularity both in research and consumer products [1–4]. However, this only applies with natural languages; barely any research exists for controlled natural languages. A project that uses speech in a controlled natural language system has been implemented by Kaljurand and Alumae [21]; they explored the properties of a controlled language when determining results from a speech interface service. This is similar to the work done in this paper, except the speech interface for this project was intended to explore the idea of an additional interface to support a CNL. In other words, Kaljurand and Alumae used a CNL to complement a speech recognition service, while this project uses a speech recognition service to complement a CNL.

Chapter 3

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 was identical to the existing prototype with regards to functionality; however, the overall user interface design will be open for changes.

3.1 User Requirements

The first type of requirements will be user requirements, which specifies what the user expects the system to do. User requirements are typically gathered from the expected users themselves; for this project, Dr Schwitter acted as the user and through meetings, we agreed upon what the system should be.

- User Requirement U.1: The user is able to enter words and form sentences without requiring to learn PENG syntax.
- User Requirement U.2: The user is able to delete characters, words and entire sentence fragments in the text input area using the backspace or delete key.
- User Requirement U.3: The user is able to add a new content word when the word is not yet in the lexicon.
- User Requirement U.4: The user is able to see the resulting answer set as soon as a new sentence has been processed.

- User Requirement U.5: The user is able to save and load specifications.
- User Requirement U.6: The user is able to use speech in addition to text as a form of input.
- User Requirement U.7: The user is able to use the menu-based interface to compose a sentence word by word.

3.2 System Requirements

The next set of requirements will be used as building blocks for when implementation begins. These are the system requirements and they can be grouped into two main categories: functional and non-functional requirements.

3.2.1 Functional Requirements

The requirements in this Section describe the intended behaviour of the system and the steps required to perform a behaviour. The behaviours of the system can be expressed as tasks, services, or functions the system is required to perform.

- Functional Requirement F.1: The system must work on any desktop computer or laptop running Mac, Windows, or Linux operating systems while using Google Chrome.
- Functional Requirement F.2: The system must update the menu interface with correct values when a token has been submitted or deleted.
- Functional Requirement F.3: The system must allow a new token to be added to the lexicon when an unrecognised token is entered.
- Functional Requirement F.4: The system must be able to save and load specifications.
- Functional Requirement F.5: The system shall have two modes of operation: text mode and speech mode.
- Functional Requirement F.6: The predictive text editor of the system must be able to operate offline.
- Functional Requirement F.7: The system must display the correct answer set programs when a new specification is submitted.

- Functional Requirement F.8: The system must allow for the reasoner mode to be set in either normal, brave, or cautious mode; the default setting being reasoner mode.
- Functional Requirement F.9: The system shall have a suggestion box to be used for auto-completion and the suggestion box shall be updated whenever a new character is entered.
- Functional Requirement F.10: The system shall update the anaphoric expressions table when a new token has been submitted.

3.2.2 Non-Functional Requirements

The non-functional requirements describe the attributes of the system such as performance, maintainability, availability, cost-effectiveness, and capacity. Unlike functional requirements that describe specific behaviours, non-functional requirements specify the criteria that will be used to determine the quality of a system when operating. The measurements required for non-functional requirements tend to require a deployed system in order to test effectively. The end results of this project was a prototype and much of the results was based on the language processor implementation, as such, tests regarding non-functional requirements were not focused on. However, some measurements were still required in order to collect the results in Chapter 7.

Chapter 4

System Architecture and Implementation

4.1 Architecture

4.1.1 Client-Server Architecture

Like the existing prototype by Nalbandian and Schwitter [19] and the web-based predictive editor by Guy and Schwitter [7], PENG^{ASP} Multimodal uses a client-server architecture. This architecture separates the workloads between the component asking for service (the client) and the provider of the service (the server). In this case, the client acts as the intermediate between the user and the server. The communication requires the client-side to initiate the communication to the server and this will cause to the server to return a JSON object based on the requested data. The web-based predictive editor that Guy and Schwitter developed, as shown in Figure 4.1, has only the client-side and server-side. This project and the one done by Nalbandian and Schwitter [19] made changes in the architecture by extending it with a speech recogniser, as shown in Figure 4.2.

4.1.2 Client-Side Architectural Pattern

4.1.2.1 Model-View-ViewModel

The largest part of the project will involve the client-side; therefore, using a design pattern will facilitate the development process. The design pattern that PENG-Speech uses is called Model-View-ViewModel (MVVM), which is a variation of the Model-View-

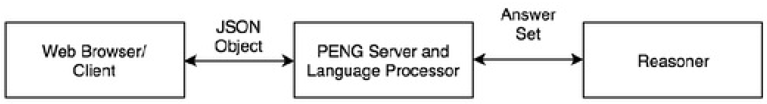


Figure 4.1: General client-server architecture with no speech interface.

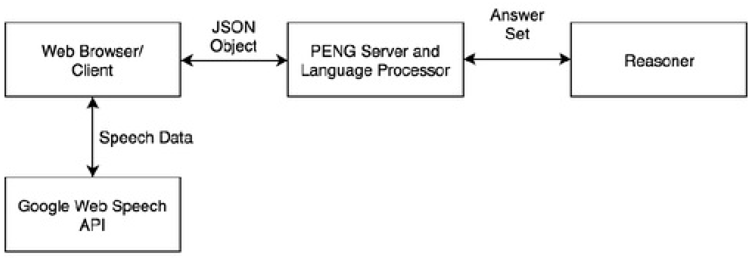


Figure 4.2: General client-server architecture with speech interface.

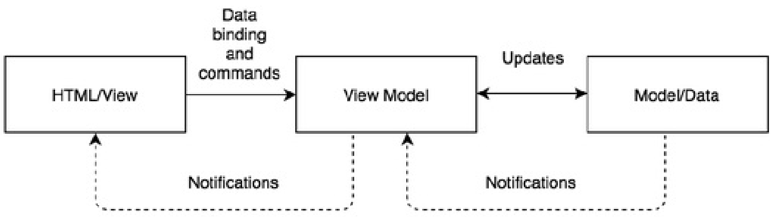


Figure 4.3: Model-ViewModel-View Diagram.

Controller (MVC) pattern [22].

MVVM [22] has three components: model, view, and view-model. The roles and responsibilities of each component serve to separate the concerns between the user interface (view) and the business logic (model). This is achieved by isolating the view and model from each other by allowing the view-model to act as the intermediary.

In the case for PENG^{ASP}, the HTML file acts as the view, the ViewModel object written in Javascript acts as the view-model, and the objects containing the data are the models; as shown in Figure 4.3.

4.2 Implementation and Tools

When investigating strategies to reduce complexity in implementation, various considerations were used as a basis to formulate a solution. The main concern was the fact that this application will be a highly responsive one; as such, the application needs to be designed in a way to minimise any performance drawbacks from being highly responsive. These drawbacks may come from the constant back and forth communication between the client and the server-side; it can also occur because of client-side rendering. Overall, we want to look at tools and implementation techniques to overcome these types of challenges.

A table containing the full list of tools used can be found in Appendix B.

4.2.1 Single Page Application

A Single Page Application (SPA) is a web-based application that uses a single HTML file containing all the components of the user interface. SPAs update the user interface dynamically without requiring continuous page loads and all in a single page. Deciding to make it a SPA was a simple choice due to the many advantages that follow it. That is, it works extremely well on highly responsive applications since the communication between data and presentation is typically handled by an intermediate layer, offline processing can be made possible for various of the functionalities, and rendering no longer needs to be done on the server side and instead can be achieved in the client-side. These are just the minor advantages of SPAs, the main advantage is that it can produce the best user experience that can be delivered in a web browser. This means web applications can be developed to be just as responsive and usable than what you typically find in native mobile or desktop applications, which is hard to achieve in cross-platform applications [23].

4.2.2 Git and GitHub

For this project, the version control Git and GitHub were used to keep various versions of the code as they progress through during implementation. Git [24] will be used to track the various changes made and Github will be used to store the changes in our repository. By tracking and storing the changes, the project will be able to mitigate the various risks involved in losses in files and reverting back to a previous state in the event an unwanted bug is introduced. Github was chosen in particular largely due to the free services it provides to registered students and its large community support [25]. In addition to its free services, it is also a great tool for collaborating and sharing code.

4.2.3 Javascript and JQuery

The largest part of the code will be written in Javascript. Javascript is a dynamically typed programming language designed for HTML and the web. Javascript was the obvious choice with its dominance as the programming language of the web and its supportability by today's web browsers. In addition to Javascript, JQuery [26] will be used to simplify implementation by making use of library functions instead of coding it from scratch. JQuery also has features such as DOM manipulation, event handling and AJAX calling that will be required throughout the implementation.

4.2.3.1 KnockoutJS

To simplify implementing the MVVM pattern, KnockoutJS [27] was an appropriate choice. KnockoutJS is a Javascript library for MVVM implementation; it complements the MVVM pattern by allowing developers to create interfaces that are in sync with the corresponding data object model (DOM). This form of DOM control allows us to scale the page by enabling specific control in data, which prevents the system from recomputing every single data, even those unrelated to the current context. Instead, its system of observables knows which part of the data is relevant and only computes those instead. In addition, it is a library and not a framework like AngularJS, Ember.js, or Backbone.js; this allows the application to play well with various other components which were an important factor in reducing implementation complexity.

4.2.3.2 Superfish

The menu-based interface for the predictive editor was implemented using Superfish [28], a jQuery plugin. This takes care of the complexity involved in implementing functionalities such as hover event handling and sub-menu collapsing. Furthermore, based on the existing system developed by Guy [7] and Schwitter [7], Superfish has demonstrated to be highly responsive for these types of applications.

4.2.4 SWI-Prolog

The backend of this application was implemented using SWI-Prolog [29]. The backend refers to the server and the language processor. The server acts as the medium between the client-side and the language processor where a JSON object from the client-side and passes it onto the language processor. The language processor examines the input and determines that it conforms with the PENG language, it then returns lookahead data, anaphoric expressions, and spelling suggestions.

The majority of the backend had already been implemented and the only changes required for this project was in the server-side. This was simply adding handlers for each of the HTML, CSS, and Javascript files.

4.2.5 Google Web Speech API

When investigating which speech interface to use, Google Web Speech API [30] was the strongest choice when looking at various factors such as cost, implementation complexity, and accuracy. The two speech interfaces that were considered was IBM Speech-to-Text [31] and Google Cloud Speech API [32]. The approach for choosing a speech interface was to make it completely free of service and Google Web Speech API provides exactly that for its speech services. Furthermore, the speech interface must also not add substantial complexity when implementing.

IBM Watson Speech-to-Text was initially the favourable choice during the planning phase. However, when looking at implementation complexity and cost, Google Web Speech API [30] seemed more appropriate. IBM Watson Speech-to-Text [31] required a charge of 0.2 USD per minute after the first thousand minutes in a month. The charge may not seem much; however, compared to the free service that Google Web Speech API provides, Google Web Speech API is the superior choice with cost efficiency. Implementing IBM Watson Speech-to-Text required a server to be running locally and implemented in Node.js. This is inferior Google Web Speech API, which can be implemented using Javascript code and without having to run a server locally on top of the PENG server.

Google Cloud Speech API had a similar pricing model to the IBM Watson Speech-to-Text where they charge 0.006 USD per 15 seconds after the first 60 minutes per month. Again, it also requires Node.js to implement as a web application. For these requirements, the Google Web Speech API was the better choice [31,32].

4.2.6 Google Chrome

The system was not built with the intention of it being cross-browser compatible due to the speech interface being only available on Google Chrome [33], therefore, using Google Chrome was absolute required.

4.2.7 Macbook Pro 2015

The implementation and experiments will be conducted using a Macbook Pro running on OSX El Capitan.

4.2.8 QuickTime Player

QuickTime Player is a built-in desktop recording software available on OSX El Capitan. It will be used to record the experiments proposed in Section 1.2.1.

4.2.9 Predictive Text Editor Implementation

The first and the most time-consuming part of the implementation was the predictive text editor. This was largely due to the number of functionalities that was required from the system. There were two main components of the predictive text editor: the first was the input area, and the second was the predictive menu-based interface.

4.2.9.1 Input Area

The input area was implemented to detect whenever a character was entered by the user and determined whether a word was ready to be sent as a token to the server. This was achieved using the following logic:

```
keyUpdate: function(data, event) {
        if (isEndOfToken(event.keyCode)) {
                this.postToken(currentToken);
                this.updateAll();
                this.currentToken = "";
        } else {
                this.currentToken += event.keyCode;
        }
}
```

`keyUpdate()` was implemented as an event handler to respond at every character input. The first statement in the method checks whether a token has been completed, this was achieved by using specific characters to indicate token completion. If a token has been completed, it sends the token to the server and updates the predictive editor with the data collected from the server. Otherwise, it updates a string variable that tracks the current token.

The `postToken()` function executes other functions that deal with any error handling and updates for the interface. In particular, it notifies the user of any error involving with incorrect input, retrieving various types of data from the server and uses them to update the interface.

```
 1
 2  postToken(token) {
 3          var request = $.when(this.post(token));
 4          request.done(function(data)
 5          {
 6                  if (isValid(token)) {
 7                          var json = JSON.parse(data);
 8                          if (this.isEndOfSpecification(token)) {
 9                                  this.updatePara(json.para);
10                                  this.updateAna(json.ana);
11                                  this.updateAnswer(json.answer);
12                                  this.updateASP(json.asp);
13                          }
14                          this.updateLookahead(json.asp);
15                  }
16                  else {
17                          this.displayError();
18                  }
19
20          });
21  },
```

The first line in the `postToken()` function calls the `post()` function which manages the formatting necessary before sending a request to the PENG server. When a response occurs, there are three scenarios that the system is prepared to respond to. The first scenario is when the token is valid and has not yet reached the end of a complete specification, it will update the lookahead information table (line 13). The second scenario is when the token is valid and has reached the end of a complete specification, it will update the lookahead information, generated paraphrases, anaphoric expressions, answers, and answer set programs. The third scenario is when an invalid token has been submitted, it will update all the relevant flags such that all the necessary precautions will be laid out before a user can proceed to another input.

Each of the variables updated or called have been declared as an observable, a property in the KnockoutJS library [27]. Declaring variables as observables handle all the changes to the user interface without being concerned about what's going on internally.

4.2.9.2 Menu Interface

The predictive menu-based interface was built along side the text area implementation using primarily Superfish. It was part of the `updateAll()` method that deals with all the updates such as lookahead data, anaphoric expressions, error handling, and various outputs for the interface. The menu had three levels: lookahead information indicator, categories, and words. The first level was the lookahead information indicator which was simply used to indicate that the menu was for lookahead information. The second level contains the available categories and appear once the user clicks or hovers on the first level. The third level contains the corresponding words from each category and only appear once the user hovers on a category.

The actual implementation for the menu interface itself was mainly done in the HTML file, as shown below:

```html

<ul class="sf-menu" id="example" >
<li class="loadlook-div" class="current">

  <a href="" data-bind="loadLookahead">Lookahead Information</a>
  <ul data-bind="foreach: lookaheadObject">
    <li class="current">
      <a href="javascript:void(0);" data-bind="text:cat"> </a>
      <ul data-bind="foreach: wfm">
        <li>
          <a href="javascript:void(0);"
          data-bind="text:$data, click: $root.postWordClicked"> </
              a>
        </li>
      </ul>
    </li>
  </ul>

</li>
</ul>
```

The code above shows the logic behind each layer of the lookahead information. All the data for the lookahead information can be found in the `lookaheadObject`. The variable `lookaheadObject` contains a list of arrays which contain the available words (wfm) for a corresponding category (cat). That is, for each available category in the `lookaheadObject`, a list of words corresponds to the category.

4.2.10 Speech Interface Implementation

The speech interface was the last major component of the implementation. The implementation of the Google Web Speech API uses the following template:

```
if (!('webkitSpeechRecognition' in window)) {
        upgrade();
} else {
        var recognition = new webkitSpeechRecognition();
        recognition.continuous = true;
        recognition.interimResults = true;

        recognition.onstart = function() { ... }
        recognition.onresult = function(event) { ... }
        recognition.onerror = function(event) { ... }
        recognition.onend = function() { ... }
        ...
}
```

The first line checks whether the current browser is Google Chrome, if it is not, a message will be displayed specifying that the current browser is not Google Chrome. In the event the browser is Google Chrome, it goes to the else statement which provides all the necessary interfaces and sets a number of attributes and event handlers. The first statement inside in the else block creates a variable for a webkitSpeechRecognition object. Next, the two attributes 'continuous' and 'interimResults' are given a boolean value of true. The 'continuous' variable determines whether to continue recognising speech during pauses. The 'interimResults' variable determines whether the speech recogniser returns back results that are definitively correct rather than only returning final results. After setting the attributes, the event handlers `onstart()`, `onresult()`, `onerror()`, and `onend()` are set. The `onstart()` event handler is triggered by clicking a button and was used to activate the speech recogniser. It then calls `onresult()`, which is an event handler that provides all the results from the speech recogniser. After that, it goes to `onevent()`, which is the event handler that stops the microphone and ends the session. Alternatively, if an error occurs, the `onerror()` event handler displays a message specifying the error.

Among those event handlers mentioned above, `onresult()` was used to update the user interface to show speech data results and to send tokens to the PENG server, as shown below.

```
recognition.onresult = function(event) {
    var interim_transcript = '';

    for (var i = event.resultIndex; i < event.results.length; ++i) {
        if (event.results[i].isFinal) {
            final_transcript += event.results[i][0].transcript;
            viewModel.postSpeechToken(final_transcript);
        } else {
            interim_transcript += event.results[i][0].transcript;
            viewModel.displaySpeechToken(interim_transcript);
        }
    }
};
```

There are two variables in the code above that are particularly worth mentioning. The first variable is `interim_transcript`, which is a string that contains the interim result from the speech recogniser. The second variable is `final_transcript`, which is a string that contains the final results from the speech recogniser. Once the speech is final, the system will send each token to the PENG server, otherwise, it will only display it on the interface.

The posting of speech token was a different process in comparison to how it was done with the text editor. The string variable `final_string` contains multiple tokens that cannot just be sent the server, instead it had to be split into single tokens and sent to the PENG server individually, as shown below.

```
1 postSpeechToken: function(words) {
2   var wordsArray = this.wordsToArray(words);
3 
4   for (var i = 0; i < wordsToArray.length; i++) {
5     this.postToken(wordsToArray[i]);
6   }
7 }
```

As shown above, the variable `words` was used to create a list (line 2) containing each individual token from the variable `words`. Each of these tokens are then sent to the PENG server where it will return the necessary lookahead data.

4.3 Development Process

4.3.1 Iterative prototyping

The project will follow the plan described in Section 1.2, where for each week, Dr Rolf Schwitter and I will agreed upon a feature or functionality to be implemented. In order to ensure project success, a workflow process was developed, as follow:

1. Discuss a feasible functionality, feature, or bug that needs to fixed.
2. Implement a working version
3. User acceptance and code review
4. Repeat

Following this process made sure that the system was being built with a reliable feedback from at every stage from an important stakeholder. Each functionality or feature implemented had been defined at a meeting and allow for errors to be detected early in the process. Only after it passes the user acceptance tests and code review, do we move on and repeat the process all over again. The testing and user acceptance will be done from a user's perspective by simulating various scenarios. As for the code review, a discussion will be conducted to discuss the overall quality of the code. The Consultation Meetings Attendance From can be found in Appendix A.

4.3.2 Programming Methodology

The organisation of the code was achieved by applying object-oriented design methodologies. This was achieved by combining the attributes (data) and procedures (methods) into single entities called objects. Object-oriented programming (OOP) was primarily used in order to improve code maintainability, which was a major concern in the existing implementation. By using OOP, the code size was significantly reduced compared to the existing system and fixing bugs did not require a lot of man-hours.

Chapter 5

PENG^{ASP} at Work

This section will demonstrate the working prototype of PENG^{ASP} and will explain the rationale behind each decision made throughout the design and implementation process of the current demonstration.

These demonstrations will be a series of images and descriptions of actions to reach the point in the image. Each demonstration will include various of the functionalities defined in the requirements document at Chapter 3, such as: entering specification via typing, entering specification via menu-interface, saving a specification, loading a specification, adding a token to the lexicon, and entering a specification via speech input.

5.1 Current State of Project

Before the demonstration, a brief overview of the PENG^{ASP}'s interface will help provide some context in these demonstrations. The general layout of PENG^{ASP}, as shown in Figure 5.2, can be divided into three components. At the top is the settings area which are the menu's that allow users to save, load, change input mode and change reasoner mode. In the middle is the user input area where the user is able to submit specifications using either text input, menu-based interface, and speech (assuming it's in text mode). The bottom component is the results area, which displays the specifications submitted, generated paraphrases, answer to queries, answer set programs, and answer sets.

5.2 Demonstration Setup

The start each demonstration, various software must be used in order for the application to compile correctly. The PENG server must be compiled using the SWI-Prolog compiler and the client-side must be running on a Google Chrome browser. In addition to the required software, Google Chrome must have its microphone permission setting to allow sites to use the microphone for the demonstration that used speech inputs.

As for the hardware used, the machine used for development (Macbook Pro 2015) had been used. In addition, each of these demonstrations had also been tested on a Lenovo ThinkPad e460e with dual boot capabilities while running Ubuntu 14.04 and Windows 10.

5.3 Input Interface Demonstrations

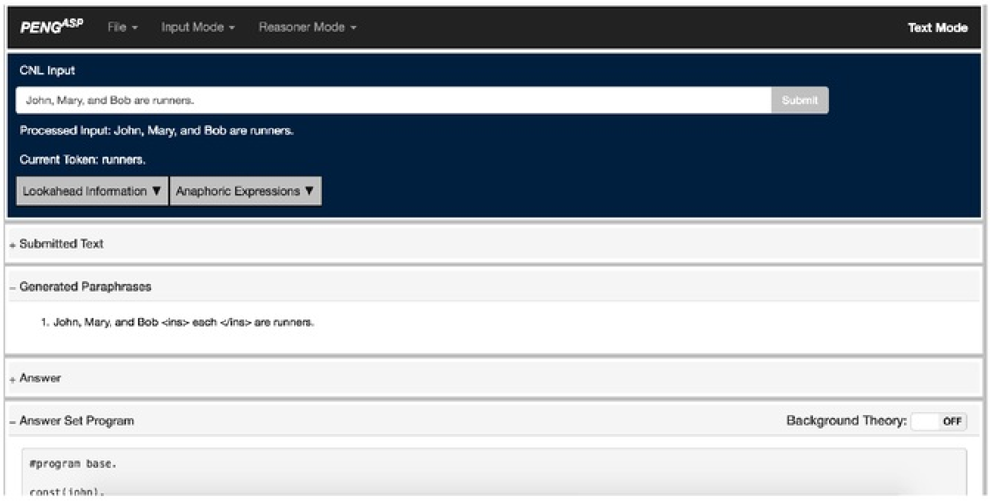
This section will demonstrate the text-based, menu-based, and speech interface. For each demonstration, PENG text will be written using one of the input methods.

5.3.1 Typed Input

This section will demonstrate the process and behaviours when writing a specification using the text editor. The image at Figure 5.1, shows when a user has entered the sentence, "John, Mary, and Bob are runners.". When typing inputs, the trigger to send a token to the server occurs when a full-stop, question mark, comma, or blank space character is detected by the key handler. For this demonstration, the system only sent tokens to the PENG server through it detecting comma's, blanks and a full-stop.

The first step was the same for all demonstrations, it must send a blank space token to the server as part of a JSON object. For this case, it was triggered by the user clicking inside an empty input area and confirming that the top of the token stack was not a blank space token. The result of this event was an AJAX response from the PENG server with data containing an ID of the last token submitted, lookahead data, and anaphoric expressions.

Next, the user entered "John," followed by a comma and a blank space; the blank space triggered the same event as earlier, however, this time, we send "John" instead of the blank space and the server returns a JSON object with the same types of data but of different values as before with the blank space. Immediately after it sends the token "John", it sends the comma token to the server and does exactly what was done with the



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.

```
#program base.

const(john).
```

Figure 5.1: Screenshot of entering specification via typing

blank space token and "John" token.

This process is repeated with the remaining words until the full stop is detected. The full stop causes the client-side to send two JSON objects to the server with different settings in reasoner mode: the first was one that contained "runners", and the second was the full-stop token. The JSON object with the "runners" token had its reasoner setting identical to what it had been to the tokens prior, which was set at "off". The JSON object with the full-stop token had its reasoner setting to "on", this only occurs when a sentence is complete and a full-stop character or question mark token was submitted.

When a full-stop has been submitted, the PENG server returns a JSON object with different types of data. The types of data it return includes id, anaphoric expressions, answer, generated paraphrases, syntax tree, and answer set program. As shown in Figure 5.1, the answer set programs are partially displayed and this was displayed due to entering a full-stop.

5.3.2 Menu-based Interface

This section will demonstrate the process and behaviours when writing a specification using the menu-based predictive interface. The image at Figure 5.2, shows when a user writes the incomplete sentence, "John works". In this case, the trigger to send a token to the server occurs whenever the user clicked on a token in the menu-based interface.

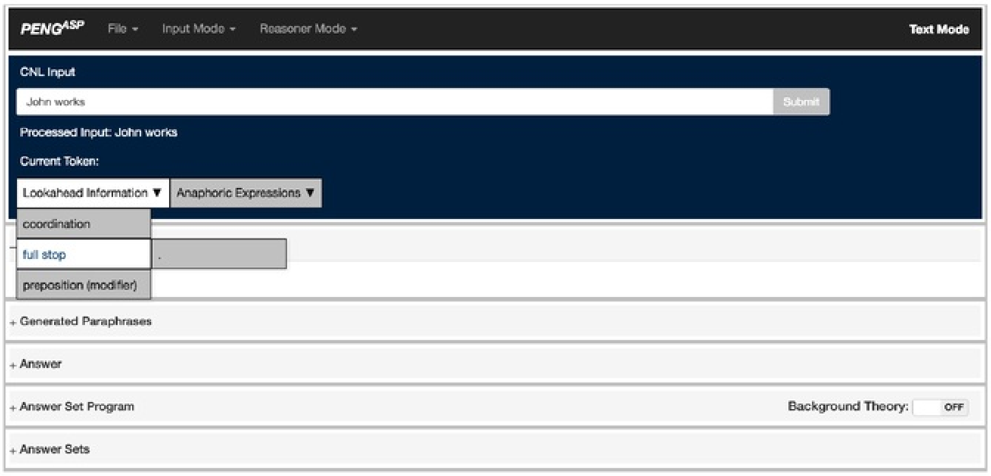


Figure 5.2: Screenshot of entering specification via menu interface

The first step begins similarly to the typed input demonstration, it starts a JSON object with a blank space as its token to the server. Although unlike the typed input demonstration, the triggering event is a mouse click on the menu labelled with "Lookahead Information". This collapses the available categories of the words that can be submitted as the next token. To display the words from the corresponding category, a simple hover on one of the categories will open a side-menu with a list of words. In this case, the user hovered on the category "name" and then selected the name "John". This process is repeated with the token "works", except the user no longer needs to do the initial click to send a blank space to open the categories menu, only a hover is needed.

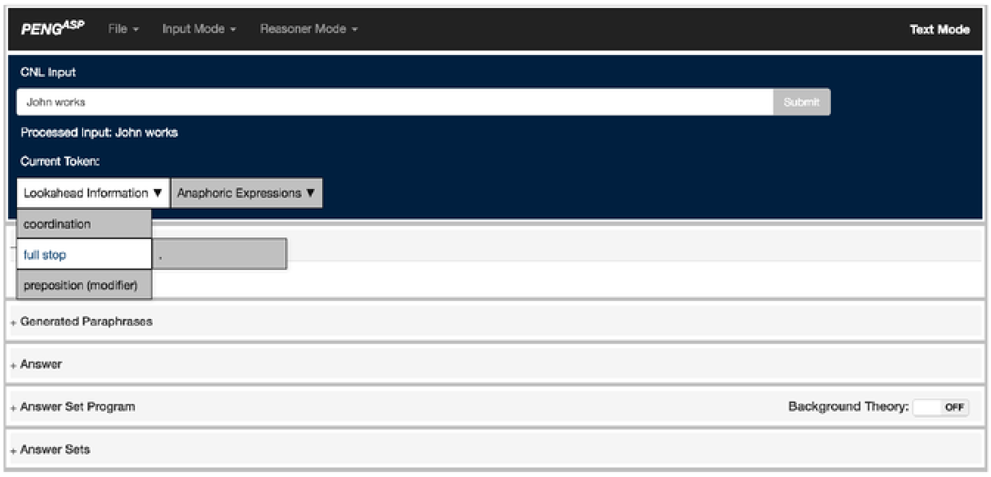
In order to take advantage of code reusability, the menu-based interface relies on the same logic used in the text input for the sending of tokens. This will also take care of determining errors and any of the predictive data required for the next set of outputs.

5.3.3 Speech as Input

This section will demonstrate the speech interface of the system. Speech is the alternative input that can work along side the various functionalities used in the previous demonstrations. For this demonstration, the user was in the middle of submitting the incomplete sentence "John and Mary are runners" using the speech interface, as shown in Figure 5.3.

To setup this demonstration, the system must be in speech mode which is activated by clicking the 'Input Mode' menu and selecting 'Speech'. This displays the microphone button. Once the microphone button has been clicked, the button begins to animate, as shown in Figure 5.3. The animated button indicates that the system is now listening for speech input. In this demonstration, the system has detected "John and Mary are runners" but have yet to process it.

Speech inputs can only processed when one of these two events occur: the first event is when the user clicks the microphone button while its listening, and the second is when a full-stop has been detected. When the microphone button is clicked while listening, the microphone shuts down the speech data gets processed. When the speech recogniser detects a full-stop, it also stops and the speech gets processed.



CNL Input: John works

Processed Input: John works

Current Token:

Background Theory: OFF

Figure 5.3: Screenshot of entering specification via menu interface

5.4 Saving, Loading, and Adding Demonstrations

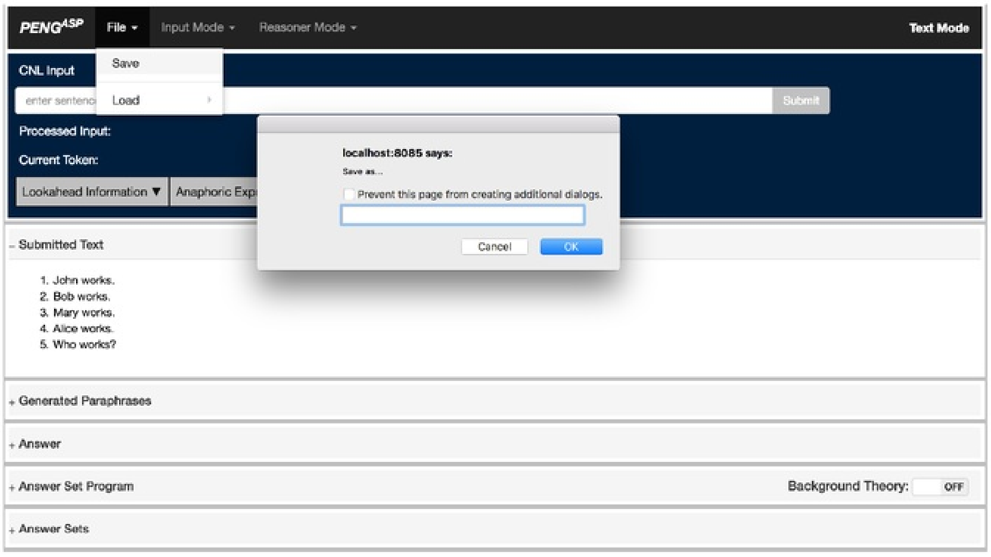
This section will demonstrate the saving (User Requirement U.5), loading (User Requirement U.5), and adding (User Requirement U.3) functionalities described at Section 3.1.

5.4.1 Saving Specifications

This section will demonstrate the saving functionality of the system. For this demonstration, four sentences have been processed, as shown in Figure 5.4, below "Submitted Text".

To save a specification, a user must first navigate their way by first clicking the 'File' drop-down menu, followed by clicking the 'Save' button. Once the save button has been clicked, a windows prompt appears with an output message, as shown in Figure 5.4. The windows prompt specifies whether the user wants to save the specifications or cancel. If the user chooses to save the specifications, the user must enter a name, otherwise it will warn the user of the error. If instead the user enters a name and saves the specification, the system will save the specifications as a text file (*.txt).

The server requires a single string to be passed as part of a JSON object. This string will be used to format each specification in the text file at a character level. Therefore, the string had to be altered to improve its readability once its been saved as a text file. This alteration mainly involved the spacing between each specifications using built-in Javascript functions 'split' and 'join' or string concatenation.



localhost:8085 says:
Save as...
Prevent this page from creating additional dialogs.

1. John works.
2. Bob works.
3. Mary works.
4. Alice works.
5. Who works?

Figure 5.4: Screenshot of specifications being saved

5.4.2 Loading Specifications

This section will demonstrate the loading functionality of the system. For this demonstration, the user has already clicked the load button on a completely empty list of specifications, as shown in Figure 5.5.

To load a specification, a user must first navigate their way by first clicking the 'File' drop-down menu, followed by the 'Load' drop-down sub-menu, and then selecting a text file to load. Once a file has been selected, it begins to load the specifications inside the

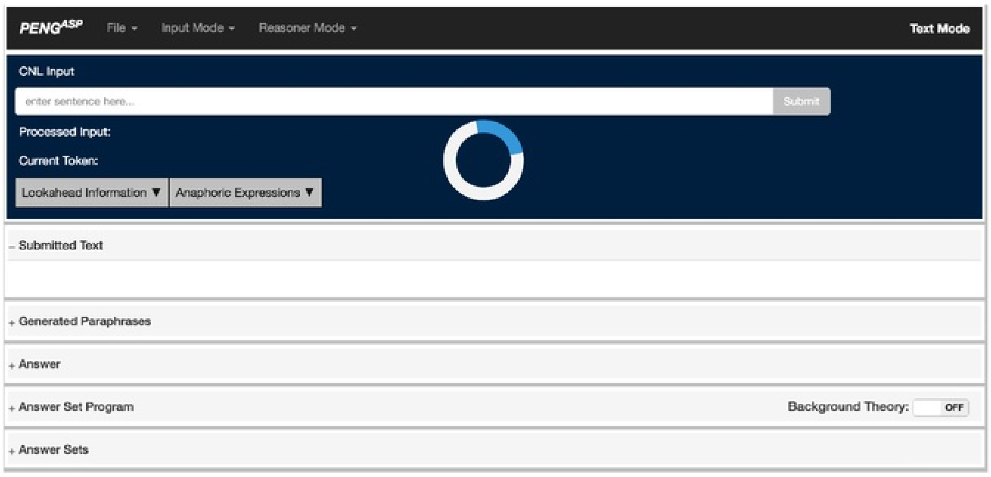


Figure 5.5: Screenshot of specifications being loaded

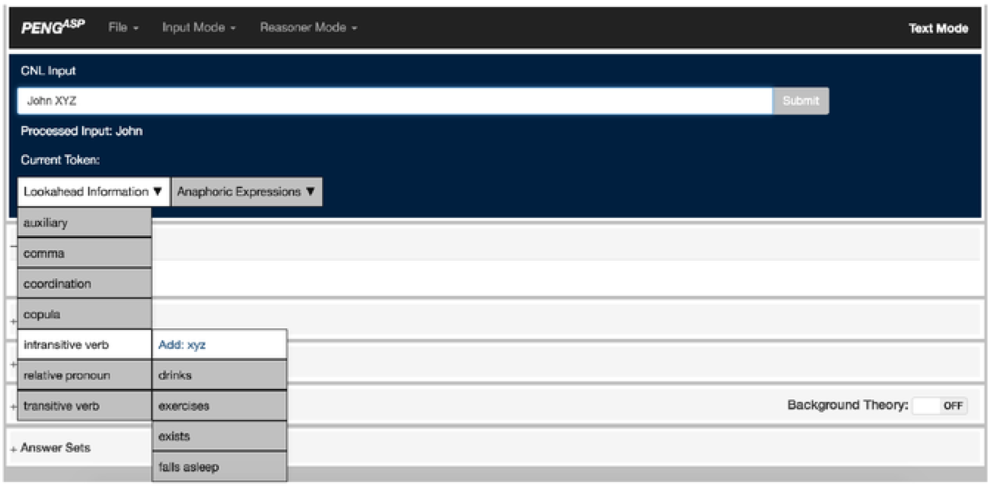
text file. The specifications must follow a specific format in order to be loaded, preferable in the same format that was outputted using the save functionality. Note that it will only load if no sentences have yet to be processed.

In the event a user attempts to load a text file while the submitted text is clear, a windows prompt appears and gives the user the option of either saving, clearing the current specification, or cancelling the load. If the user decides to save the specification, it immediate clears the specification and loads the text file. If the user decides to clear the specification, it does not save the specification and instead loads it after being cleared.

5.4.3 Adding Token to Lexicon

This section will demonstrate the adding functionality of the system. The adding functionality occurs when an unrecognised token had been recognised and will allow the user the option of making it recognisable to the system. For this demonstration, the user is attempting to adding 'Alice XYZ' in the user lexicon, as show in Figure 5.6.

To setup this demonstration, the word being added must not be recognisable to the system at this stage of the specification. In this case, the token 'XYZ' had not been recognised, we can see this as 'XYZ' is clearly shown in the input text area, but does not show in the 'Processed Input' area. To add 'XYZ' to the lexicon, the user must use the 'Lookahead Information' menu interface and navigate their through an appropriate category and select the first item in that list. Assuming that it is an appropriate category, the first will be of the format 'Add: ⟨*word*⟩', where ⟨*word*⟩ is replaced with the unrecognised word, or in this case, 'XYZ'. The adding of unrecognised words is only reserved for content words.


CNL Input

John XYZ

Processed Input: John

Current Token:

Background Theory: OFF

Figure 5.6: Screenshot of a token being added to the lexicon

Chapter 6

Experimental Setup

This chapter describes the setup used to collect the data from the experiments conducted for this project. The purpose of these experiments was to investigate whether or not the project goals mentioned in Section 1.1 had been satisfied. That is, determining the practicality of speech input in a CNL system, comparing the new system with the existing prototype [19], and comparing Google Web Speech API with IBM Watson Speech-to-Text.

6.1 Practicality of Speech Setup

This section will look at the procedure and data required for determining the practicality of a speech interface in CNL system. Measuring the practicality of the speech interface required collecting data related to the interaction between a user and the system. The collection of data came in two forms: accuracy and input duration time. The input user time refers to the duration for a user to enter sentences using one of them system's input interfaces.

6.1.1 Accuracy For User Input

Measuring the accuracy of the speech interface was done by observing the number of incorrect tokens detected during the process of constructing PENG sentences via speech input. Only those that were sent to the PENG server was considered as incorrect tokens.

6.1.2 Input Duration Time

The input duration time refers to the number of seconds it takes for a user to construct a list of PENG sentences. This experiment was conducted using each of the three input interfaces (text, menu, and speech) individually and using the same list of PENG sentences for each test.

6.1.3 Data and Procedures

As mentioned, the types of data collected (accuracy and duration) will require different types of observation. The accuracy looks at the number of wrongs words submitted, while the duration looks at the how long it takes a user to input sentences into the system. Collecting both types of data for the speech interface did not require the experiments to be done individually. Instead, each input method was simulated five times, where each time the accuracy and duration was recorded for the speech interface.

6.1.3.1 Apparatus

In order to make the observations, the desktop recording (specified in Section 4.2.8) software, QuickTime Player, was used. Using a desktop recorder was a useful tool for collecting data for these experiments. By video recording the experiment, the number of errors and the duration was easily captured. In addition to a desktop recorder, a variable was created in the code to keep track on the number of times an incorrect token was submitted.

For each experiment, the system was running locally. Running it locally required a machine to have SWI-Prolog in order to compile the server. In addition to using SWI-Prolog (specified in Section 4.2.4), the system was using a Google Chrome browser (specified in Section 4.2.6).

A list of sentences written in the PENG language was used for all experiments. The list of sentences can be found in Appendix C; it contains seven sentences and required 74 tokens to be processed by the server to complete.

6.1.3.2 Procedure

This section will describe the procedure used for each input method. Although the input methods differ, the general approach for each experiment was to keep it as identical as possible. As such, the PENG text to be entered for each experiment was the same. The experiment will span out into five sessions to avoid any noticeable improvements in user performance. Each session, a type of input cannot be experimented more than a single time.

The ordering for which interface was experimented on at each session was the following:

Session 1:	Text Input	$\longrightarrow$	Menu Interface	$\longrightarrow$	Speech input
Session 2:	Speech Input	$\longrightarrow$	Text Input	$\longrightarrow$	Menu Interface
Session 3:	Menu Interface	$\longrightarrow$	Speech Input	$\longrightarrow$	Text Input
Session 4:	Text Input	$\longrightarrow$	Menu Interface	$\longrightarrow$	Speech input
Session 5:	Speech Input	$\longrightarrow$	Text Input	$\longrightarrow$	Menu Interface

The experiment for the text-based interface will simply be creating sentences using a keyboard. Each time a sentence was completed, the user pressed the enter key to submit the sentence. The timer to measure the duration will start from when the user clicks on the input area to when the last sentence had been submitted and processed. For this experiment, only the user input speed was measured due to the typed input being dependent on a user's ability to type. In addition, PENG^{ASP} Multimodal enforces correct form at a token level, therefore, a measurement of accuracy was unnecessary.

The experiment for the menu-based interface will be creating sentences using only the menu interface. Each time a sentence was completed, the user will click the 'Submit' button to submit the sentence. For this experiment, only the duration will be measured due to the menu-based interface enforcing correct form. The duration will start from when the user clicks the 'Lookahead Information' menu to when the last sentence has been submitted and processed.

The experiment for the speech interface will be creating sentence using only speech input. Like the procedure for the menu interface, a complete sentences will be submitted by using the 'Submit' button. The timer to measure the duration will start from when the user starts the microphone to when the last sentence has been submitted and processed.

6.2 System Reliability Comparison Experimental Setup

This section will describe the tools and procedures used in order to collect the data that compared the new system with the existing system. In order to compare the two systems, experiments were done to obtain data relating to reliability.

6.2.1 Measuring Reliability

The measuring of system reliability was determined by its duration before failure; failure in this context refers to when the system crashed and required the PENG server to be restarted. This will be achieved by simulating functionalities at random; this includes using all the input interfaces.

6.2.2 Data and Procedures

To measure the reliability, the same materials and procedures that had been used for the speech interface experiment were used. In addition to using all input interfaces, the general functionalities such as saving and loading a specification and adding unrecognised tokens to the lexicon.

To make sure that no advantage was given due to a difference in hardware performance, the experiments had all been done using a Macbook Pro. In addition, aside from the screen recorder, all background processes in the workstation was terminated.

6.3 Speech Interface Comparison Experiment

This section will describe the data and procedures to compare Google Web Speech API and IBM Speech-to-Text. In order to compare the two speech interfaces, experiments and research were done to obtain results relating to its accuracy in recognising speech. However, it is important to note that the accuracy in recognising speech was dependant on the ability of the speaker; in this case, the subject was a native English speaker.

6.3.1 Data and Procedure

This section will describe the data and procedures to compare Google Web Speech API and IBM Speech-to-Text. In order to compare the two speech interfaces, experiments and research were done to obtain results relating to its accuracy in recognising speech. However, it is important to note that the accuracy in the recognising of speech was dependent on the ability of the speaker; in this case, the subject was a native English speaker.

The experiment used three different lists of sentences and recorded each one to be tested. The three lists can found in Appendix C, D, and E.

Chapter 7

Results

This chapter presents the results obtained from the experiments described in Chapter 6; that is, the practicality of a speech interface and a comparison of the two systems. To present the results, tables were created to represent each type of data. In addition to the tables and accompanying text were used to comment on any observations made during the experiments.

7.1 Practicality of Speech

This section presents results from the experiment regarding the practicality of speech input in a CNL system. The data produced for this experiment included the accuracy and speed during user input, where the three types of input methods (text, menu, and speech) were tested.

7.1.1 Accuracy

The accuracy results in Table 7.1 are presented in three columns. The first column contains the test number for each of the tests. The second column contains the number of errors counted for a corresponding test. The third column contains the accuracy rate relative to the number of tokens there was in the text file '07-successful-student-a.txt' for each test. At the time of the experiment, the text file contained 74 tokens.

7.1.1.1 Accuracy of the Speech-based Interface

The highest accuracy rate came in test number 4 with a rate of 98.65%. The highest accuracy rate came in test number 5 with a rate of 94.60%. Resulting in an average rate of 96.49%.

Accuracy: Speech Input		
Test Number	Number of Errors	Accuracy Rate
1	3	95.95%
2	2	97.3%
3	3	95.95%
4	1	98.65%
5	4	94.6%

Table 7.1: The number of incorrectly submitted to the PENG server and the accuracy rate via speech input.

7.1.2 Input Duration Time

The duration during user input results in Table 7.2, 7.3, and 7.4 all have three columns. The first column contains the test number. The second column contains the duration (in seconds) of a test while entering the sentences in the text file '07-successful-student-a.txt'. The third column contains the number of tokens sent per second. The time per sentence was given through duration (column 2) by the number of sentences in the text file.

7.1.2.1 Input Duration Time: Typed

The shortest time recorded occurred in the second test at 95 seconds. The longest time recorded occurred in the fifth test at 119 seconds.

Duration and tokens per second: Text Input		
Test Number	Duration (seconds)	Tokens per second
1	101	0.73
2	95	0.77
3	116	0.64
4	97	0.76
5	119	0.62

Table 7.2: The durations for entering the sentences in 07-successful-student-a.txt via text input.

7.1.2.2 Input Duration Time: Menu-based Interface

The shortest time recorded occurred in the second test and finished at 388 seconds. The longest time recorded occurred in the first test and finished at 466 seconds.

Duration and tokens per second: Menu-based Input		
Test Number	Duration (seconds)	Tokens per second
1	456	0.16
2	388	0.19
3	460	0.16
4	466	0.15
5	414	0.18

Table 7.3: The durations for entering the sentences in 07-successful-student-a.txt via menu-based input.

7.1.2.3 Input Duration Time: Speech Interface

The shortest time recorded occurred in test 1 and finished at 52 seconds. The longest time recorded occurred in test 5 and finished at 152 seconds.

Duration and tokens per second: Speech Input		
Test Number	Duration (seconds)	Tokens per second
1	52	1.42
2	77	0.96
3	132	0.56
4	81	0.91
5	152	0.47

Table 7.4: The durations for entering the sentences in 07-successful-student-a.txt via speech input.

7.1.2.4 Text, Menu, and Speech Duration During User Input Results Comparison

This section will present the average duration and tokens per second for the five experiments of each input method. In addition, the difference between each of the average results of the three interfaces will be presented.

DURATION		
	Duration (Average)	Tokens per second (Average)
Text	105.6s	0.70
Menu	436.8	0.17
Speech	98.8	0.75
Difference between speech and text-based interface	6.8s	0.05%
Difference between speech and menu-based interface	338	0.58%

Table 7.5: The average durations for entering the sentences in 07-successful-student-a.txt and differences compared to speech input.

7.2 System Reliability Results

This section presents the results from the experiments which compared the reliability of PENG^{ASP} to the system built by Nalbandian and Schwitter [19]. The data for the reliability was the duration before system failure.

7.2.1 Reliability

The results in this Section was presented in Table 7.6 and Table 7.7. For each experiment, the duration and the reason for failure will be presented.

7.2.1.1 PENG^{ASP} Reliability

The shortest time before the system had crashed using the PENG^{ASP} system was 52 seconds. The longest time before system failure had occurred at 482 seconds. In addition to the duration, the main reason to why it failed was due to the inputs not corresponding with the token being sent to the server.

7.2.1.2 The Existing System's Reliability

The shortest time before the system had crashed using Nalbandian and Schwitter's system was 13 seconds. The longest time before system failure had occurred at 217 seconds. For 3 of the experiments, the system had failed when the use of backspace was required.

Duration and reason for system failure (PENG^{ASP} Multimodal)		
Test Number	Duration (s)	Reason
1	482	Incorrect speech input detected
2	52	Incorrect text input detected
3	417	Incorrect speech input detected
4	431	Backspace bug
5	164	Incorrect text input detected

Table 7.6: The durations before system failure and the reasons for failure using PENG^{ASP}.

In addition to the duration, the main reason for failure was due to the inputs not corresponding with the tokens being sent to the PENG server. For those with 'Unknown' in the 'Reason' column, this was due to the unfamiliarity with the implementation and no known cause could be made.

Durations and reasons for system failure (Existing System)		
Test Number	Duration (s)	Reason
1	46	Backspace bug
2	13	Backspace bug
3	112	Unknown
4	217	Backspace bug
5	91	Unknown

Table 7.7: The durations before system failure and the reasons for failure using the existing system.

7.2.1.3 Comparing Reliability Results

In average, Nalbandian and Schwitter's system crashed at 29.77 seconds, while PENG^{ASP} Multimodal crashed in average at 309.2 seconds. PENG^{ASP} Multimodal in average was able to last 279.43 seconds longer than the system developed by Nalbandian and Schwitter.

Comparison of average durations before system failure	
Test Number	Duration (Average)
PENG^{ASP}	309.2 seconds
Nalbandian and Schwitter's System	29.77 seconds
Difference	279.43 seconds

Table 7.8: Average reliability results and differences of both systems.

7.3 Speech Interface Comparison Results

This section presents the results collected from the speech interface comparison experiment. To present the results, the tables have four columns. The first column contains the name of the text file read. The second column contains the number of words read aloud. The third column contains the number of incorrect words found for a corresponding test. Finally, the fourth column contains the accuracy rate relative to the number of words read aloud, specified in each table.

In addition, a table containing the average results for all the files loaded will be calculated and the differences between the two speech interfaces will be calculated.

7.3.1 Google Web Speech API Accuracy Results

When using Google Web Speech API, the highest accuracy rate occurred for the file `07-successful-student-a.txt`, while the lowest accuracy rate occurred for the file `36-brave-cautious-reasoning-1.txt`. The output results for this experiment can be found in Appendix F, H, and J.

Accuracy: Speech Input (Google Web Speech API)			
File name	Number of words in file	Number of incorrect words	Accuracy Rate
07-successful-student-a.txt 60	4	5	93.33%
31-lrec-2.txt	57	4	92.98%
36-brave-cautious-reasoning-1.txt	37	3	91.89%

Table 7.9: The number of incorrect words using Google Web Speech API.

7.3.2 IBM Speech-to-Text Accuracy Results

When using IBM Speech-to-Text, the highest accuracy rate occurred for the file `07-successful-student-` while the lowest accuracy rate occurred for the file `36-brave-cautious-reasoning-1.txt`. The output results for this experiment can be found in Appendix G, I, and K.

Accuracy: Speech Input (IBM Speech-to-Text)			
File name	Number of words in file	Number of incorrect words	Accuracy Rate
07-successful-student-a.txt	4	5	93.33%
31-lrec-2.txt	57	15	73.76%
36-brave-cautious-reasoning-1.txt	37	10	72.97%

Table 7.10: The number of incorrect words using IBM Speech-to-Text.

7.3.3 Average Accuracy Results Comparison

In average, Google Web Speech API had an accuracy rate of 92.73%, while IBM Speech-to-Text had an accuracy rate of 80.02%. The Google Web Speech API was 12.71% more accurate than the IBM Speech-to-Text.

Accuracy: Speech Input (IBM Speech-to-Text)	
Speech Interface	Accuracy Rate (Average)
Google Web Speech API	92.73%
IBM Watson Speech-to-Text	80.02%
Difference	12.71%

Table 7.11: Average accuracy results and difference of both systems.

Chapter 8

Discussion

This chapter provides an explanation and evaluation for the results presented in Chapter 7. It will review the findings of our research and discuss the various limitations discovered during our investigation.

8.1 Evaluating the Practicality of Speech Input

To do an evaluation for the practicality of speech input, two types of data were collected: the accuracy and the input duration time. At this stage, it is not clear what the statistical difference would have been given that a larger variety of test subjects had participated. Based on this, we can only assume that these results are accurate for subjects that are native English speakers.

8.1.1 Accuracy Evaluation

Overall, using the speech interface in the PENG^{ASP} system resulted in quite a high accuracy rate, indicating that the words had been easily recognised and/or the subject had performed the experimental task with a clear speech. The average accuracy rate was 96.49%, which is a respectable result; however, this can vary quite significantly depending on whether the subject is a non-native or native English speaker. It is not clear whether using a speech interface would produce better accuracy results than the text-based interface due to it being based on the user's typing ability.

8.1.2 Duration During User Input Evaluation

The results for the during duration of user input experiment showed that on average it takes a user 98.8 seconds to enter 74 tokens using speech as input. The same experiment also showed that in average it takes 105.6 seconds using the text-based interface and 436.8 using the menu-based interface to enter the same 74 tokens. This gives average rates of 0.75 tokens per second for speech, 0.7 tokens per seconds for text-based, and 0.17 tokens per second for a menu-based interface. The average results show that the speech interface was the fastest method for user input, while the menu-based interface was the slowest. The difference between the text-based interface and the speech interface was quite insignificant with the speech interface being only 0.05% better. However, when looking at the slowest times (Table 7.4) for the speech interface, we observed that the accuracy (Table 7.1) has a significant effect on the duration of the speech interface.

8.1.3 Overall Evaluation of Speech Interface in a CNL System

The standard input method for most CNL systems has been text-based. When comparing the speech interface with the text-based interface, our investigation suggests that in average the speech interface was the faster input method. However, when looking at test numbers 3 & 5 for Table 7.1 and Table 7.4, the time increases significantly when the number of errors increased. However, when no errors were made, it was significantly faster than all input methods. Given these findings, the speech interface has demonstrated to be the most optimal mode of input, given that none or very little errors are made during input.

8.2 System Reliability Comparison Evaluation

As mentioned in Section 6.2, PENG^{ASP} Multimodal was compared with the existing system developed by Nalbandian and Schwitter [19]. The reason for the comparison was due to PENG^{ASP} Multimodal being a reconstruction of the system previously developed by Nalbandian and Schwitter [19]. The existing system required a reconstruction due to it being extremely unstable and difficult to maintain. In this evaluation, the main focus will be on the system's reliability.

The results from our experiment showed that PENG^{ASP} Multimodal was significantly more reliable than Nalbandian and Schwitter's system. The more reliable system refers to the system with the longer duration before system failure. For 4 out of the 5 experiments, PENG^{ASP} Multimodal produced better results than the existing system. The average results had also produced better results when using PENG^{ASP} Multimodal; with a difference of 279.43 seconds. The existing was extremely unstable and crashed when either attempting to delete a character or for unknown reasons. PENG^{ASP} Multimodal had also crashed, however, significantly less than the existing system. Overall, PENG^{ASP} Multimodal was a more reliable system due to the significant difference in the average duration before system failure.

8.3 Speech Interface Comparison Evaluation

Overall, IBM Watson Speech-to-Text performed significantly worse than Google Web Speech API. For the text file `07-successful-student-a.txt`, the two interfaces had the same results, however, after that file, the accuracy results for IBM Watson Speech-to-Text noticeably drop. No real assumptions could be made on the weaknesses of each interface due to test size; however, for this experiment, the Google Web Speech API was superior when it came to recognising names.

In average, Google Web Speech API was 12.81% more accurate than IBM Watson Speech-to-Text. Based on these results, and the reasons mentioned in Section 4.2.5, Google Web Speech API was selected as the speech interface for this project.

Chapter 9

Conclusion

In this paper, we have presented the architecture and implementation of the PENG^{ASP} Multimodal system. This system is a CNL based system with a multimodal interface. The multimodal aspect of the system consists of three input interfaces: text-based, menu-based, and speech-based. At the time of the project, CNL based systems were generally implemented with only text-based input capabilities. For this project, a speech interface was integrated into a predictive text editor. When integrating the speech interface, one of the aims was to find a solution that allowed for it to coordinate well with the text-based interface and the menu-based interface. The approach we had taken was to build the speech interface on top of the predictive text editor and make use of its functionalities. By following this approach, the development time was reduced and the system was able to enforce the form of the PENG language into the speech interface, namely, Google Web Speech API.

Towards the end of this thesis, a study was conducted to determine the practicality of speech input in a CNL based system. Prior to this research, it had yet to be done. To gather the results of the study, experiments were conducted to determine the accuracy rate and the duration for a user to enter sentences written in PENG text. The findings suggest that using speech input can have significant benefits when looking at the speed in user input; however, the speed significantly declines as the number of errors grows. Overall, the decision to design the PENG^{ASP} system with a multimodal interface is a favourable approach, as it provides an additional option for user input and can resolve the unreliable aspects in speech recognition.

Chapter 10

Future Work

In Chapter 8, the practicality of speech input in CNL based system was evaluated. In this chapter, a brief outline on possible future research, beyond the scope of this project was discussed.

10.1 Usability Tests

This project had its own form of usability testing when determining the practicality of speech input; however, it only used a single test subject to determine its usability. This can be problem with speech input due to the various variables that can significantly change a speech recognisers results. For instance, our findings suggested that the number of errors significantly declined the speed of user input using speech. If that was the case, the results would be much more different with a non-native speaker uses the system, as opposed to a native speaker.

The experiments conducted for this project had focused on the user input time; however, no consideration on the actual comfort level of a user was ever factored in. To do this, various types of users will need to be tested using speech as input in CNL system and measured based on their user experience. This will allow for stronger findings in determining the practicality of speech input in a CNL system.

10.2 CNL Speech Interface

One of the major limitations this project faced was its inability to control specific parameters in the speech interface. The parameters that would have improved the system significantly was limiting the words detectable by the speech recogniser. For example, if the word such as those that homophone in nature, such as "flour" was detected, but only "flower" was in our lexicon, the speech recogniser should be able to infer that "flour" was intended. This can be further developed by creating a speech interface specifically for a controlled natural language. This type of speech interface would have produced much more accurate results and may potentially increase the average speech input speed.

Chapter 11

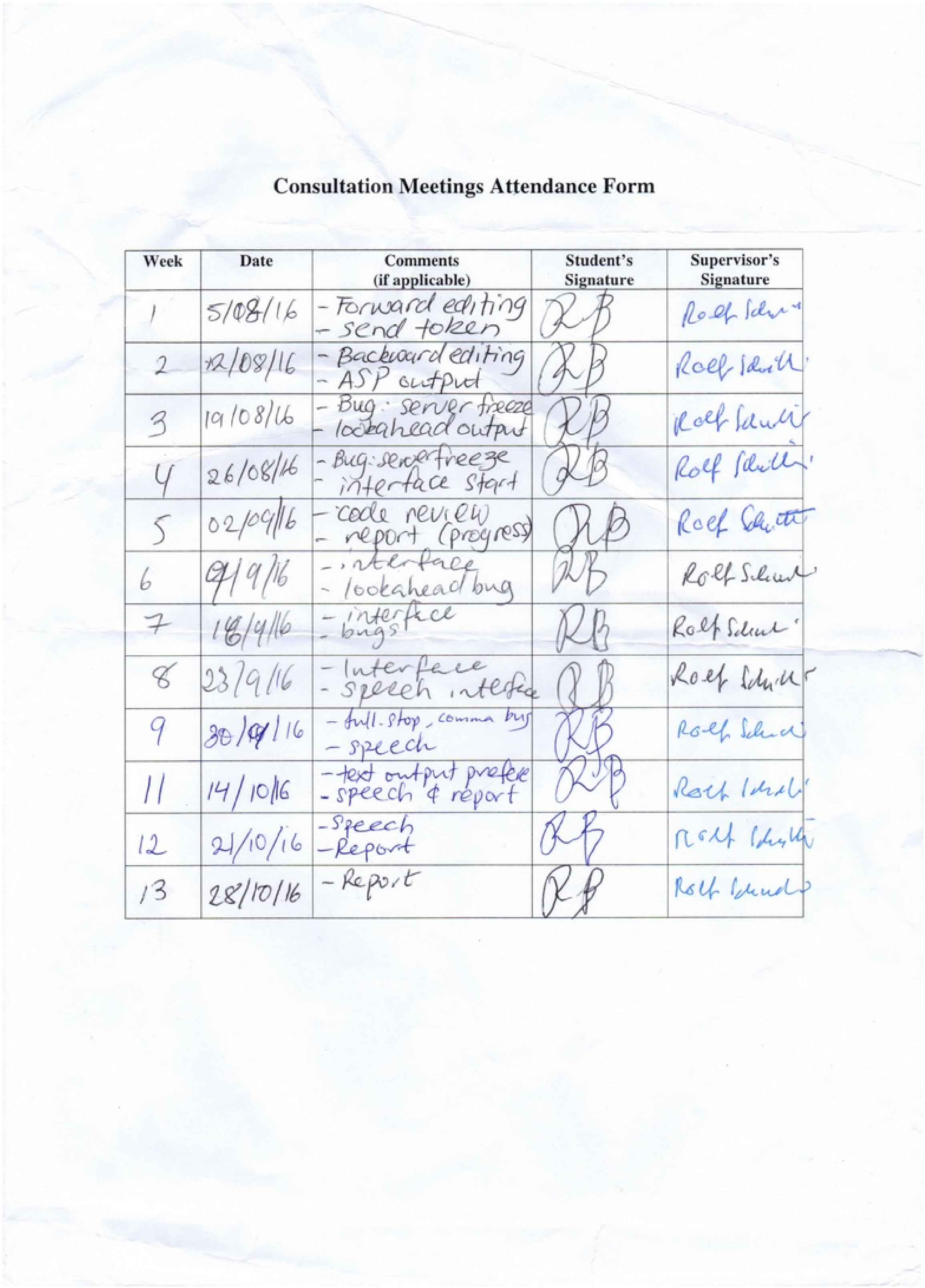
Abbreviations

CNL	Controlled Natural Language
PENG	Processable English
ASP	Answer Set Programming
API	Application Programming Interface
ACE	Attempto Controlled English
CPL	Computer Processable Language
AJAX	Asynchronous JavaScript And XML

Appendix A

Consultation Attendance Meeting Form

The Consultation Attendance Meeting Form signed by Dr Schwitter and I as required by the department of engineering.



Consultation Meetings Attendance Form

Week	Date	Comments (if applicable)	Student's Signature	Supervisor's Signature
1	5/08/16	- Forward editing - send token	RB	[illegible]
2	12/08/16	- Backward editing - ASP output	RB	[illegible]
3	19/08/16	- Bug: server freeze - lookahead output	RB	[illegible]
4	26/08/16	- Bug: server freeze - interface start	RB	[illegible]
5	02/09/16	- code review - report (progress)	RB	[illegible]
6	9/9/16	- interface - lookahead bug	RB	[illegible]
7	16/9/16	- interface - bugs	RB	[illegible]
8	23/9/16	- Interface - speech interface	RB	[illegible]
9	30/09/16	- full-stop, comma bug - speech	RB	[illegible]
11	14/10/16	- text output prefere - speech & report	RB	[illegible]
12	21/10/16	- Speech - Report	RB	[illegible]
13	28/10/16	- Report	RB	[illegible]

Appendix B

Tools List

The table contains the versions and specifications of the software and hardware used.

Software & Hardware		
Name	Version	Specification
Git	2.8.1	Mac OSX Build
Javascript	1.8	N/A
JQuery	1.8.17	N/A
KnockoutJS	3.3.0	N/A
SuperFish	1.7	N/A
SWI-Prolog	7.3.24	64-bits, Mac OSX build
Google Web Speech API	N/A	N/A
Google Chrome	54.0.2840.71	64-bits, Mac OSX build
Macbook Pro	Retina, 13-inch, Early 2015	3.2GHz, Intel Core i7, 16GB RAM
QuickTime Player	7.7.9	Mac OSX v10.6.3 or later Build

Appendix C

Text file: 07-successful-student-a.txt

```
Every student who works is successful.

Every student who studies at Macquarie University works or
parties.

It is not the case that a student who is enrolled in Information
Technology parties.

Tom is a student.

Tom studies at Macquarie and is enrolled in Information Technology.

Bob is a student who studies at Macquarie and does not work.

Who is successful?
```

Appendix D

Text file: 31-lrec-2.txt

```
Dave and Olivier are lecturers.

John is a student who is enrolled in Mathematics.

Thelma is a student and is enrolled in Mathematics.

Every lecturer supervises exactly two students who are
enrolled in Mathematics.

Dave, Olivier and Ron are lecturers.

For every student who is enrolled in Mathematics there
are exactly two lecturers who supervise the student.
```

Appendix E

Text file: 36-brave-cautious-reasoning-1.txt

```
Every media company owns one or more newspapers.

Fairfax Media Limited is a media company.

The Sydney Morning Herald, The Age and The
Australian are newspapers.

Fairfax does not own The Australian.

Which newspapers does Fairfax own?
```

Appendix F

Google Web Speech API Results: 07-successful-student-a.txt

```
Every student who works is successful.

Every student who studies at Macquarie University works or parties.

It is not the case that a student who is enrolled in information
technology parties.

It is a student.

Studies at Macquarie and is enrolled in information technology.

Bob is a student who studied at Macquarie and does not work.

It successful?
```

Appendix G

IBM Watson Speech-to-Text Results: 07-successful-student-a.txt

```
Every student who works it successful.

Every student who studies at Macquarie university works or parties.

It is not the case that a student enrolled in information
technology parties.

Paul is a student.

Pardon studies at Macquarie bank is enrolled in information technolog

Bob is a student who studies at Macquarie and does not work.

Who is successful?
```

Appendix H

Google Web Speech API Results: 31-lrec-2.txt

```
Dave and Oliver are lectures.

John is a student who is enrolled in mathematics.

Belmore is a student and is enrolled in mathematics.

Every lecturer supervisors exactly two students who are enrolled
in mathematics.

Dave, Oliver and Ron are lecturers.

For every student who is enrolled in mathematics the exactly two
letters to supervise a student.
```

Appendix I

IBM Watson Speech-to-Text Results: 31-lrec-2.txt

```
They and all over our lectures.

John is a student Houston role in mathematics.

Belmont is a student Anderson rolled in mathematics.

Every lecturer supervises exactly two students who are enrolled
in mathematics.

Dave Oliver and wrong our lectures.

For every student Houston rolled in mathematics there are exactly
2 lectures.
```

Appendix J

Google Web Speech API Results: 36-brave-cautious-reasoning-1.txt

```
Every media company owns one or more newspapers.

Fairfax Media limited media company.

Sydney Morning Herald, The Age and the Australian
are newspapers.

Fairfax does not own the Australian.

Which newspaper does Fairfax own?
```

Appendix K

IBM Watson Speech-to-Text Results: 36-brave-cautious-reasoning-1.txt

```
Every media company owned one or more newspapers.

Fairfax media limited is a media company.

The Sydney morning Herald. H. M. do you spell yet
our newspapers.

Fairfax does not own deeds failure.

Which newspaper does Fairfax owed?
```

Bibliography

[1] J. Hirschberg, "Advances in natural language processing," *Science*, 2015.

[2] "Use siri on your iphone, ipad, or ipod touch," Apple Inc., Accessed on: 3-11-2016. [Online]. Available: https://support.apple.com/en-au/HT204389

[3] "Now," Google Inc., Accessed on: 1-11-2016. [Online]. Available: https://www.google.com/search/about/learn-more/now/

[4] "Cortana - meet your personal assistant," Microsoft Corporation, Accessed on: 2-11-2016. [Online]. Available: https://www.microsoft.com/en-au/mobile/experiences/cortana/

[5] K. S. Jones, "Natural language processing: a historical review," *Artificial Intelligence Review*, 2001.

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

[7] S. Guy and R. Schwitter, "Architecture of a Web-based Predictive Editor for Controlled Natural Language Processing," *Language Resources and Evaluation*, pp. 1–26, 2015.

[8] T. Kuhn, "AceWiki: Collaborative Ontology Management in Controlled Natural Language," *In Proceedings of the 3rd Semantic Wiki Workshop, CEUR Workshop Proceedings*, 2008.

[9] T. Kuhn and R. Schwitter, "Writing Support for Controlled Natural Languages," *Controlled Natural Language: Workshop on Controlled Natural Language*, 2009.

[10] T. Kuhn, "A Survey and Classification of Controlled Natural Languages," *Computational Linguistics*, pp. 121–170, 2015.

[11] C. Ogden, "Basic English: A general Introduction with Rules and Grammar," 1930.

[12] G. Getto, “Simplified Technical English: Specification ASD-STE100,” AeroSpace and Defence Industries Association of Europe, 2013, Accessed on: 3-11-2016. [Online]. Available: http://guiseppegetto.com/pwr393/wp-content/uploads/2013/02/ASD-STE100-ISSUE-6.pdf

[13] “Simplified Wikipedia,” Wikimedia Foundation, Accessed on: 05-10-2016. [Online]. Available: https://simple.wikipedia.org/wiki/Main_Page

[14] N. E. Fuchs, K. Kaljurand, and T. Kuhn, “Attempto Controlled English for Knowledge Representation,” *Reasoning Web*, vol. 5224, pp. 104–124, 2008.

[15] 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.

[16] 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

[17] R. Schwitter, “Processing coordinated structures in PENG light,” *Advances in Artificial Intelligence*, pp. 658–667, 2011.

[18] 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.

[19] R. S. Christopher Nalbandian, “A Speech Interface to the PENG ASP System,” pp. 48–57.

[20] H. R. Tennant, K. M. Ross, R. M. Saenz, C. W. Thompson, and J. R. Miller, “Menu-Based Natural Language Understanding,” pp. 151–158, 2011, Accessed on: 3-11-2016. [Online]. Available: http://www.aclweb.org/anthology/P83-1023

[21] K. Kaljurand and T. Alumae, “Controlled Natural Language in Speech Recognition Based User Interfaces,” *In Proceedings of the 3rd Semantic Wiki Workshop, CEUR Workshop Proceedings*, pp. 79–94, 2012.

[22] “The mvvm pattern,” Microsoft Corporation, Accessed on: 11-10-2016. [Online]. 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, Accessed on: 6-12-2016. [Online]. 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] J. Resig, "JQuery," Accessed on: 04-09-2016. [Online]. Available: https://jquery.com/

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

[28] J. Birch, "jQuery Superfish Menu Plugin," Accessed on: 04-09-2016. [Online]. Available: https://plugins.jquery.com/superfish/

[29] "SWI Prolog." [Online]. Available: http://www.swi-prolog.org/

[30] G. Shires, "Voice Driven Web Apps: An Introduction to Web Speech API," Google Inc., Accessed on: 04-09-2016. [Online]. Available: https://developers.google.com/web/updates/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API

[31] "Speech to Text," Internation Business Machines Corporation, Accessed on: 04-09-2016. [Online]. Available: https://developers.google.com/web/updates/2013/01/Voice-Driven-Web-Apps-Introduction-to-the-Web-Speech-API

[32] "Cloud Speech API Beta: Speech to text conversion powered by machine learning," Google Inc., Accessed on: 08-08-2016. [Online]. Available: https://cloud.google.com/speech/

[33] "Chrome for Desktop - Google," Google Inc. [Online]. Available: https://www.google.com.au/chrome/browser/desktop/