Using Text Classification To Automate Ambiguity Detection in SRS Documents

H M Ishrar Hussain

A Thesis

in

The Department

of

Computer Science &

Software Engineering

Presented in Partial Fulfillment of the Requirements

For The Degree of Master of Computer Science

Concordia University

Montreal, Quebec, Canada

August 2007

Your file Votre référence
ISBN: 978-0-494-34442-2
Our file Notre référence
ISBN: 978-0-494-34442-2

Abstract

Using text classification to automate ambiguity detection in SRS documents

H M Ishrar Hussain

Software Requirements Specification (SRS) is one of the most important artifacts produced during the software development lifecycle. In practice, requirements specifications are initially written in natural language, which allows them to be corrupted with different forms of ambiguity that eventually may contribute to critical failure in the subsequent phases of the system's development, if they are not detected at the time of requirements validation. The objective of this work is to study possible automation of detecting ambiguity in SRS documents by means of a text classification system. The work is a part of a larger project aimed at applying Natural Language Processing (NLP) techniques to assess the quality of SRS documents.

In the absence of a standard annotated corpus, we collected SRS samples and carried out corpus annotation process to build corpora of our own, one at the sentence-level and the other at the discourse-level. The annotators were trained with an annotation guideline, which was written based on the quality model that we had developed. The process showed substantial level of inter-annotator agreement indicating the possibility of automating the task by a tool that can accurately emulate the human annotation process. The resultant corpus was then used for training and testing our text classification system.

We set the scope of this thesis to detect ambiguity at the level of surface understanding only, since the indicators of its possible existence in text can be realistically extracted by the currently available NLP tools. We developed two different decision-tree-based text classification systems that worked at the sentence-level and the discourse-level, and conducted a series of experiments training and testing the classifier with different sets of features. Finally, merging the two classifiers together yielded optimum results with an accuracy of 86.67% in detecting ambiguity at the level of surface understanding. To our knowledge, none of the previous work in the field of Requirements Engineering (RE) has

tested the applicability or performance of using a text classification system to automate the detection of textual ambiguities. Our work, thus, provided significant evidence on the prospect and feasibility of using a text classifier to automate ambiguity detection in SRS documents.

Keywords

Software Requirements Specification, Text Classification, Natural Language Processing.

Acknowledgements

I pay my sincere gratitude to all the people who made this thesis possible. Much of my appreciations go to my supervisors, Dr. Olga Ormandjieva and Dr. Leila Kosseim, for their continuous guidance and support. Also, their efforts in the corpus annotation work for this thesis is invaluable.

Many thanks to the members of TROMlab and CLaC laboratory for their time-to-time suggestions, and to Shadi Moradi Seresht, for her valuable efforts in the corpus annotation. I am indebted to them all.

Finally, on a personal note, I would like to convey my thanks to my parents, and my wife for their inspirations and encouragements to complete this task.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

One of the integral phases of the software development lifecycle is Requirements Engineering (RE), where it is imperative to understand the software in whole before actually building it. The task can be cumbersome, since requirements are recorded from the stakeholders (especially clients and users), who have inadequate know-how to specify them unambiguously, often leading to an erroneous analysis. Failure in this phase can disrupt the work of any subsequent phase, causing a huge escalation in the overall cost of the software's development. A study by The Standish Group [1995] shows that the United States spend "$250 billion each year on IT application development of approximately 175,000 projects, and 31.1% of those getting cancelled before they ever get completed." The study then reveals "incomplete requirements and specifications" as one of the primary reasons of their failure, and suggests "clear" requirements statement as one of the key factors of successful software development. We can, therefore, deduce that (semi-) automated detection of unclear or "ambiguous" requirements statements can serve a great deal in performing faster and more accurate Requirements Analysis, and thus, minimizing the overall costs of software development.

Our research aims to detect the presence of possible ambiguity in software requirements specification (SRS) documents during the requirements elicitation phase. Our approach relies on the usage of a text classifier that emulates human reading comprehension considering a number of quality characteristics to differentiate between ambiguous and unambiguous requirements. The work is a part of a larger project aimed at applying Natural Language

Processing (NLP) techniques to the RE process (see Figure 1). The objective of NLP assessment in the context of that project can be expressed in terms of three main goals:

G1. Automatic NLP-driven quality assessment of the textual requirements in the requirements gathering and elicitation phase.

G2. Automatic NLP-driven quality assessment of the textual requirements in the analysis and specification phase, where conceptual static and dynamic models are developed from the textual requirements.

G3. Graphical visualization and animation of the conceptual models extracted from the requirements text for the user's validation and feedback.

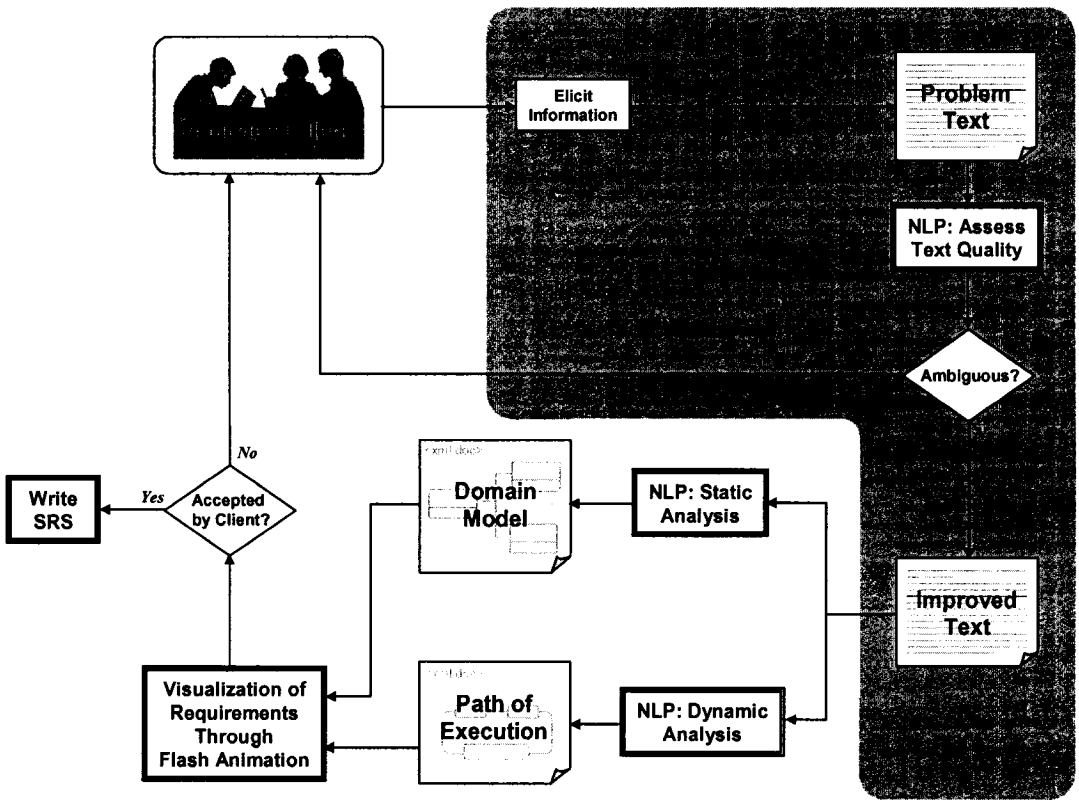

Figure 1: NLP-driven Quality Assessment in Requirements Engineering
(shaded region shows the scope of this thesis)

This thesis is concerned with the challenges inherent in understanding the initial textual requirements (as stated in G1 above) using NLP and, specifically, text classification

technique. The objective is to identify the textual ambiguities in the requirements elicitation phase before the conceptual modeling of the requirements begins. In this chapter, we will briefly introduce the concepts related to our work.

1.2 What is SRS

Software requirement specification (SRS) documents are the medium used to communicate user's requirements to technical people responsible for developing the software. Leffingwell and Widrig [2003] defined a software requirement in their book as follows:

- *A software capability needed by the user to solve a problem to achieve an objective*
- *A software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documentation.*

This indicates that the task of writing requirements as a two-way process, where both the users (and/or clients) and the technical people (analysts, developers, managers and others) are involved. Thus, the common practice is to write the SRS document without any formalization, i.e. in plain natural language (NL), so that it can be easily conveyable between the two parties. An SRS document specifies features of a software that enriches the understanding of the developers on the particular software, and also acts as the primary contract for the potential users to verify if the software meets their need. This imposes additional emphasis on its text to be clear and accurate.

1.3 Quality of SRS

An SRS document is carefully written in natural language with the goal of maintaining some good quality characteristics. In this section, we discuss the quality characteristics put forth by IEEE, and also the problems faced by a requirements specifier in maintaining these qualities.

1.3.1 IEEE Standard 830-1998

This standard describes the practices recommended by IEEE [1998] to write an SRS document, and also defines the quality characteristics of a "good" SRS document. They are: (1) Correct, (2) Unambiguous, (3) Complete, (4) Consistent, (5) Ranked for importance, (6) Verifiable, (7) Modifiable and (8) Traceable. Here, the definition of "Unambiguous" set by the standard corresponds to an SRS document, where each of its statements has only one interpretation. The IEEE standard further mentions that the inherent ambiguous nature of the natural language can make the text of an SRS document fail to comply with the aforesaid rule making it ambiguous, and thus, degrading the overall quality of the document.

1.3.2 Meyer's Seven Sins

Bertrand Meyer [1985], in his paper, showed areas of SRS document, where a specifier is more prone to make mistakes. His study presented a thorough description of such mistakes by classifying them into seven distinct categories or "seven sins", as he preferred to call them. All these sins deteriorate the quality of an SRS document. Meyer defined them as below:

Noise	*The presence in the text of an element that does not carry information relevant to any feature of the problem.*
Silence	*The existence of a feature of the problem that is not covered by any element of the text.*
Over-specification	*The presence in the text of an element that corresponds to a feature of the problem but to features of a possible solution.*
Contradiction	*The presence in the text of two or more elements that define a feature of the system in an incompatible way.*
Ambiguity	*The presence in the text of an element that makes it possible to interpret a feature of the problem in at least two different ways.*
Forward Reference	*The presence in the text of an element that uses features of the problem not defined until later in the text.*
Wishful Thinking	*The presence in the text of an element that defines a feature of a problem in such a way that a candidate solution cannot realistically be validated with respect to this feature.*

Table 1: Meyer's "The seven sins of the specifier" [Meyer, 1985]

In Table 1, we also find Meyer defining the problem of "Ambiguity" similarly to the IEEE Standard [IEEE, 1998]. We can, therefore, extend the notion in light of the IEEE standard by mentioning that the ambiguous characteristic of the natural language is responsible for this problem.

However, in this thesis, we, inspired by more recent work in this area, use to the word "Ambiguity" within a larger context (see the next section), and all "the seven sins", including what Meyer defined as Ambiguity, become subsets of what we define here as Ambiguity.

1.4 Ambiguity in SRS

Ambiguity in SRS document can exist in various forms within the NL texts of a Software Requirements Specification (SRS) document. Consider the following two examples:

(1) *Most commonly, the orders are matched in price/time priority: The order with the better price has a higher priority than an order with a worse price.* [Layda, 2000]

(2) *Design a program that allows a network operator to plan changes in every parameter of every cell in the network. These planned changes should be verified for consistency and correctness and then applied to the network in the least disturbing way.* [Kriens, 1996]

In example (1), it is impossible to interpret exactly how much price is "better" or "worse", since there can be many subjective interpretations of better or worse prices. In addition, the text also does not explain how much change the "priority" is to experience with the increase or decrease of price. Thus, example (1) is ambiguous since we cannot find the exact meaning of its text, in other words, we can come up with more than one interpretation.

On the other hand, in example (2), the text does not specify how to verify the changes of the cells for consistency or correctness. So, we find example (2) to be ambiguous as well, since the feature it describes cannot be realistically implemented in a formal design model, let

alone executable code, due to lack of information. In Table 1, Meyer defined such a phenomenon as "Silence".

Note that, like the "Silence" shown in example (2), all the "sins" of Table 1, can make the text ambiguous in such a way that it cannot be realized into any design model of formal specification. Such ambiguity usually exists within a passage, comprised of more than one sentences or paragraphs. Thus, one sentence alone may not be enough to instigate such ambiguity. On the other hand, the kind of ambiguity introduced in example (1) is limited to the scope of a single sentence, affected by linguistic elements that generate ambiguity, such as the words "better", "higher" or "worse", as shown in the example.

In this thesis, we define ambiguity as the difference between the depiction of an informal textual description of the problem (requirement) and the description of the solution for the informal domain where the intents lie. We affirm that lowering the level of ambiguity in the textual requirements document will lead to a better quality conceptual description (model) of the solution, and also reduce the amount of time required for requirements analysis and specification.

1.5 Ambiguity in terms of Reading Comprehension

In our work, we use the term ambiguity in its general sense — the characteristics of the language used in an SRS document that make its content difficult to interpret or to realize into a design model of formal specification (e.g. the Domain model, the Use Case model, System Sequence Diagrams etc.). This leads us to focus on human reading comprehension of SRS documents, which is also the foremost task in the common practice of requirements analysis.

Comprehension of the requirements text describing a problem and its domain can typically be divided into two broad levels: the literal meaning (or surface understanding) and the interpretation (or conceptual understanding). Reading at the surface level, means understanding the facts stated in the document. It allows us to answer common questions involving who, what, when and where. The second reading level involves the interpretation of the document: understanding what is meant or implied rather than what is stated. This

involves making logical links between facts or events, drawing inferences and trying to represent the content more formally. Stakeholders of all kinds involved in the use and development of a system should be able to understand an SRS at both the surface and the conceptual levels.

The following sections map the concepts of reading comprehension with the definitions of different kinds of ambiguity introduced earlier in section 1.3 and 1.4.

1.5.1 Ambiguity in Surface Understanding

In the context of our work, we use the term "Surface understanding" to denote how easy or how difficult it is to understand the facts stated in the document, without judging its design or implementation concerns in terms of any software engineering concept. Thus, increasing difficulty in this level of understanding represents possible failure to identify the exact interpretation of the text, which partly supports for what Mayer's [1985] work and the IEEE Standard [IEEE, 1998] defined as "Ambiguity".

Our previous example (1) in section 1.4, demonstrates ambiguity at the level of surface understanding. Several surface factors can be involved at this level; e.g. length of the sentences, ambiguous adjectives and adverbs, passive verbs, etc.

1.5.2 Ambiguity in Conceptual Understanding

On the other hand, we use the term "Conceptual understanding" to denote how much a developer or a requirements engineer would gain in designing or implementing a system by simply reading/examining its problem texts carefully, i.e. without applying any formalism procedure of requirements engineering. This level of understanding involves deeper factors as introduced by the "seven sins" (*Noise*, *Silence*, *Over-specification*, *Contradiction*, *Ambiguity*, *Forward Reference* and *Wishful Thinking*) described by Meyer [1985] (see Table 1 for definitions). Our previous example (2) in section 1.4, shows ambiguity at the conceptual understanding level, as it suffers from *Silence*.

1.6 Goals of the Thesis

The research described in this thesis is concerned with the challenges inherent in understanding the initial textual requirements (see G1, section 1.1) using the text classification. As stated in section 1.1, our target is to identify the textual ambiguities in the requirements elicitation phase before the conceptual modeling of the requirements begins. We consider that the root cause of errors being introduced into the requirements (and the consequent reduction in quality) is ambiguity in the text (also see our definition of ambiguity in section 1.4).

Goal 1. *To develop a quality model for requirements quality assessment derived from the existing guidelines in the literature for writing SRS documentation (such as [Fabbrini et al., 2001; Kamsties et al., 2001; Gnesi et al., 2005; Meyer, 1985; Wilson, 1997; Wilson et al., 1996]) and from the experts' experiences.*

Our quality model targeting the automatic assessment of textual requirements in terms of their ambiguity is shown in Figure 2.

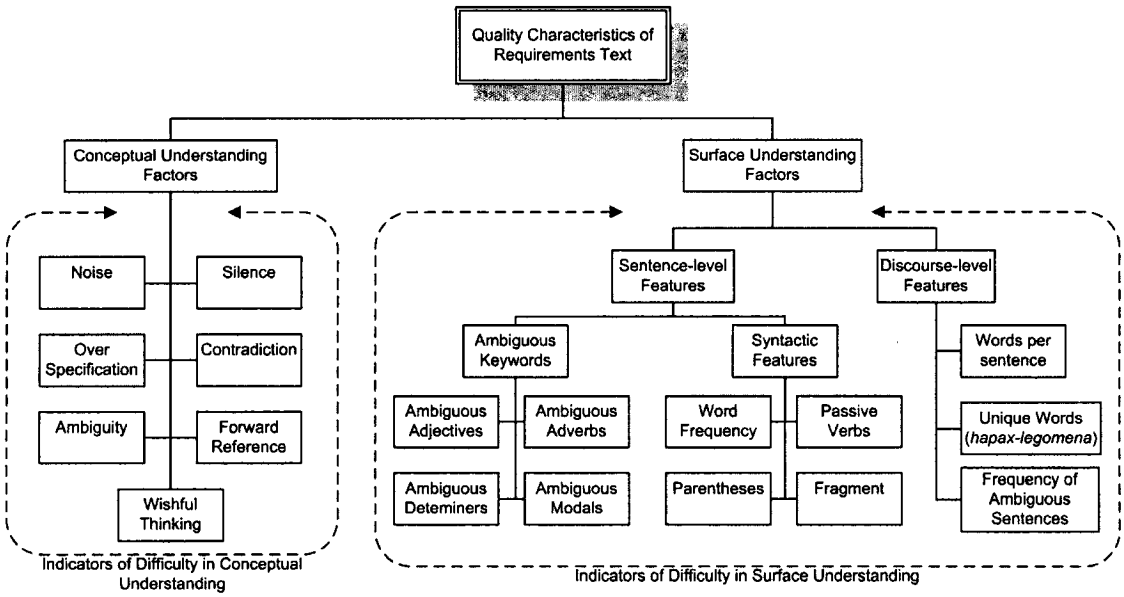

Figure 2: Quality Model for Software Requirements Specification

As mentioned in section 1.5, our model analyzes the quality of the requirements text from two different points of view, namely surface (literal) understanding and conceptual (modeling) understanding. The model presents a hierarchical decomposition of software requirements

text quality into measurable characteristics that can be collected directly from the natural language text.

The quality characteristics influencing conceptual understanding are all the 'seven sins' that Meyer [1985] mentioned (see Table 1 for details), while the characteristics influencing surface understanding are mostly lexical and syntactic features. This leads to our next goal:

Goal 2. *To investigate if it is difficult to detect ambiguity manually in terms of both surface and conceptual understanding using these quality characteristics.*

Our premise was that, if humans agree statistically on the quality of requirements texts, then the quality model truly measures what it is supposed to measure; namely, the quality of the textual description of the requirements. By contrast, if the human annotators cannot statistically agree on a classification, then the automation would be difficult to achieve and it would not be possible to evaluate the results of the automatic classification. The results are sufficient for us to believe that an automatic system can be built to emulate the decision-making process of the human annotators and to automatically classify requirements documents.

Goal 3. *To build a text classification system that could classify SRS text as "ambiguous" or "unambiguous", in terms of surface understanding, and to evaluate its performance.*

Although our ultimate target was to build a classifier that can classify a discourse in terms of its ambiguity, we focused on building a similar classifier at the sentence level first to assess the achievability of automating the task of ambiguity detection at the more limited scope of the sentence. We have determined the discriminating power of the surface understanding indicators, and have developed a classifier to actually flag ambiguous and unambiguous discourse at the surface level.

To our knowledge, this is the first attempt in the literature to apply a text classifier to software requirements quality assessment. We believe that, with proper training, such a text classification system will prove to be of immense benefit in detecting ambiguities in a software requirements text.

1.7 Overview of the Thesis

In next chapter, we present our objectives in detail and discuss our hypothesis and the research methodology. In chapter 3, we review most of the important related work published in the literature. Chapter 4 presents our work on corpus annotation and its results. Chapter 5 and 6 give detailed description of the development of the classifier, along with different experiments on it and a comprehensive analysis of their results. Chapter 7 contains a brief overview on implementation details of our system. In chapter 8, we present a critical discussion on the significance and limitations of our work. And, finally, chapter 9 contains our concluding remarks and discussion of our future work.

Chapter 2

Objectives and Research Methodology

2.1 Introduction

We have already explained the goals of this thesis briefly in section 1.6 of the previous chapter. In this chapter, we will elaborate on these goals setting a strictly defined set of objectives. We will then explain our research hypothesis and our research methodology in details. The chapter will then present a brief overview of the tools and technique used in our thesis.

2.2 Objectives

As discussed in chapter 1, the primary goal of this thesis was to identify the textual ambiguities in the requirements elicitation phase before the conceptual modeling of the requirements begins, and to study a possible automation of detecting ambiguity in SRS documents in terms of reading comprehension. We set the following objectives to achieve our goal:

A. Evaluation of human performance

Our initial objective focused on the evaluation of human performance in detecting ambiguity. To our knowledge, no one has attempted a formal evaluation of their results and a comparison to human evaluations. For this purpose, four human annotators were asked to annotate passages and sentences from 25 randomly chosen

SRS documents, each from a different domain, based on their ambiguities. Their performance was measured on the degree of their agreement with each other. Our study (explained in chapter 4 in details) evaluated the feasibility of such a task by analyzing how difficult it really is to perform and how the automatic tools developed can compare to human performance.

B. Applicability of Text Classification

Our next goal was to study the applicability of using text classification techniques in detecting ambiguity in SRS documents. Our study on the currently available tools in the field reveals that all of the 'features' (or clues) that the annotators look for in the text to annotate a passage of an SRS document as "ambiguous" in terms of surface understanding can be realistically extracted by available tools of in the field of Natural Language Processing. Conversely, the features influencing ambiguity in conceptual understanding of the SRS documents required domain knowledge and deeper semantic analysis, for which the available tools are inefficient. We, therefore, focused our objective to develop the system for detecting ambiguity at the level of surface understanding only.

C. Performance evaluation of the system

Our final objective was to build the classification system that detects ambiguity at the level of surface understanding, and evaluate its performance by comparing its results with the ones of human annotation.

2.3 Research Hypothesis

We understand that none of the previous work has tested the applicability or performance of using a text classification system to automate the detection of textual ambiguities in SRS documents. Thus, our research hypothesis was that a text classification system, trained with and tested against human annotated samples, can best emulate the human decisions taken while detecting ambiguity in the SRS documents.

To test our hypothesis, we systematically built two sets of annotated corpora — one composed of 165 annotated samples of passages, and the other with 1211 annotated samples of sentences, all from 25 randomly chosen SRS documents of different domains. The corpus was then used for both training and testing a text classification system that we later developed. Our methodology is explained in details in the following section.

2.4 Research Methodology

Our overall research methodology can be explained in the following phases:

(1) Inception

(2) Pilot Study

(3) Corpus Annotation

(4) Features Selection and Extraction

(5) The Sentence Classifier (development, training and testing)

(6) The Discourse Classifier (development, training and testing)

Figure 3 shows the summary diagram of our research methodology. The following sections explain our research methodology:

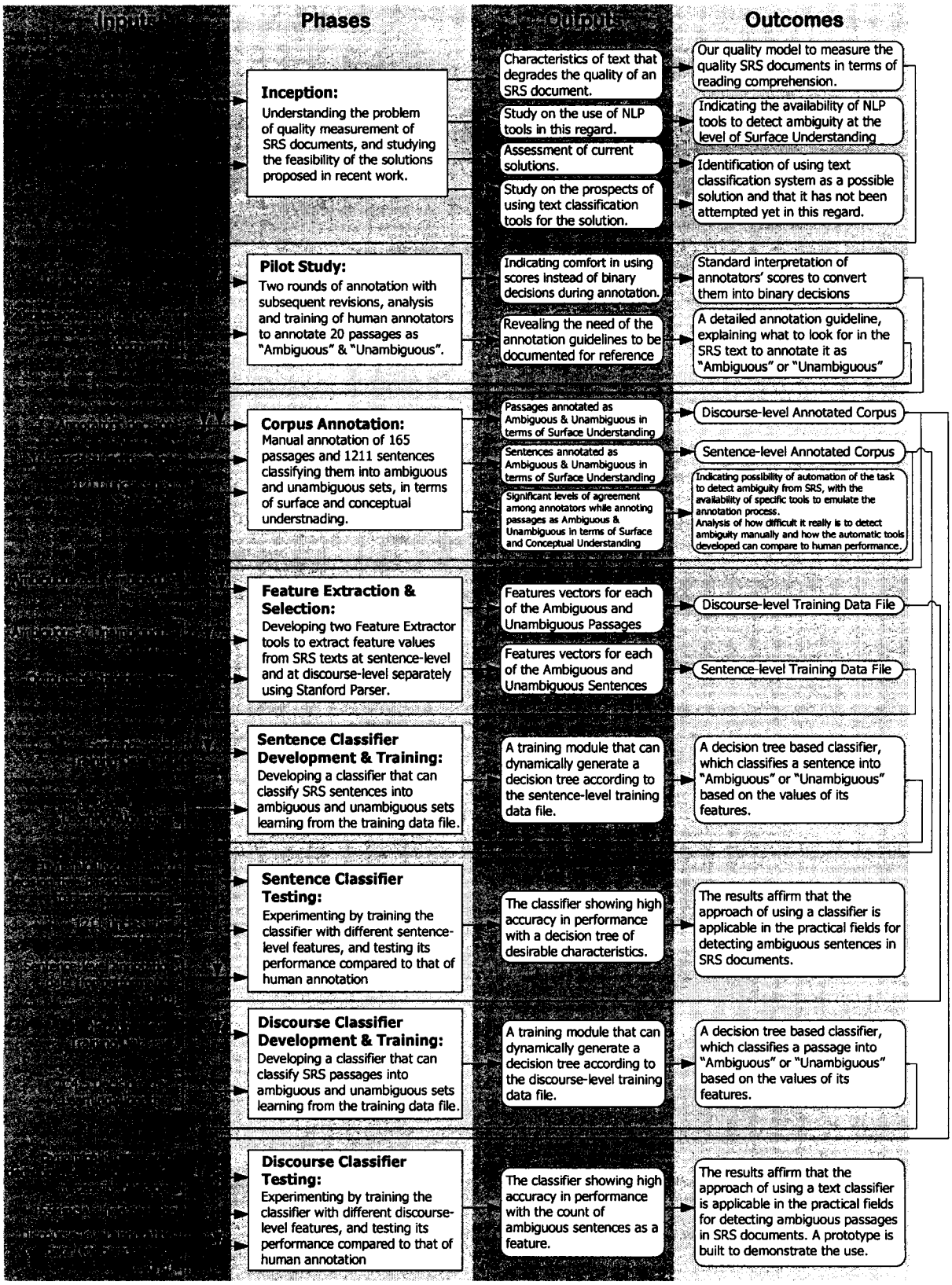

Figure 3: Summary of Research Methodology

2.4.1 Inception

Our target in this initial phase was to understand the problem of quality measurement of SRS documents, and to study the feasibility of the solutions proposed in the recent work in the related field. We went through the documentation of studies that have addressed the problem of detecting ambiguities in natural language requirements specification. We discussed these publications in details in chapter 3. We then considered our experts' opinion in choosing the attributes of the NL text having influence in the quality of SRS documents. As described in chapter 1, our idea was to derive a quality model first for requirements quality assessment from the existing guidelines in the literature for writing SRS documentation (such as [Fabbrini et al., 2001; Kamsties et al., 2001; Gnesi et al., 2005; Meyer, 1985; Wilson, 1997; Wilson et al., 1996]) and from the experts' experiences. The model [Ormandjieva et al. 2007] consisted of characteristics of natural language text that degrades the quality of an SRS document. Figure 2 in section 1.6 shows the model.

Our study identified the potential of using specific NLP tools as resources in detecting the presence of those quality characteristics that can have an influence on the surface understanding of an SRS document. For conceptual understanding of an SRS document, however, the study showed the inadequacy of NLP resources at present to detect the presence of Meyer's 'seven sins' in the text. The process of automating the detection of Meyer's seven sins requires some extent of domain knowledge and deeper semantic analysis, and currently available NLP tools are inefficient for the task. This pointed out the fact that automating the process of detecting ambiguity at the level of conceptual understanding straight from the SRS text remains still out of our reach. So, our proposed idea is to make the process semi-automated, where the system will first try to formalize an SRS document by generating formal specifications and visualization of the specification through animations. All these fall outside the scope of this thesis (see the shaded region in Figure 1). Thus, we concentrate on building the text classification system to detect ambiguity at the level of surface understanding only. Conceptual understanding was still considered separately while performing corpus annotation (see chapter 4), to understand human's performance in detecting ambiguities at both the levels of surface and conceptual understanding.

Our study in this phase also revealed that, to this date, no research has attempted to use a text classification technique in detecting ambiguity of SRS documents. This emphasizes the importance of our research results in the RE field.

2.4.2 Pilot Study

Before conducting the actual annotation process, we carried out two rounds of annotation with subsequent revisions, analysis and training of human annotators to annotate each of the 20 passages from our corpus as "Ambiguous" or "Unambiguous". This not only familiarized our annotators with the annotation process, but their experiences also prompted us to devise a scoring technique (described in section 4.4.1.3, in details). It also revealed the need to state an annotation guideline, which defined a lenient set of rules for the annotators to understand and perform the process of annotation.

2.4.3 Corpus Annotation

We asked the annotators to annotate two different corpora: one holding 165 passages and the other 1211 sentences, all from 25 randomly chosen SRS documents, each from a different domain. The annotation process was conducted to classify these passages and sentences into two categories: ambiguous and unambiguous, in terms of surface and conceptual understanding. The annotated passages were used for creating the discourse-level corpus, while the annotated sentences were used for creating the sentence-level corpus. There have also been significant levels of agreement among the annotators while annotating passages as ambiguous and unambiguous in terms of both surface and conceptual understanding. This affirms the possibility of automating the task of detecting ambiguity with the availability of practical tools that emulate the human annotation process. The data also helped us analyze how difficult it actually is to detect ambiguity manually, and how the automatic tools developed can compare to human performance. See chapter 4 for the results and analysis in details.

In addition, the analysis in this phase indicated a positive correlation between the surface and conceptual understanding of the text, and a negative correlation between the understanding and the time required to analyze a text. The above confirmed our premise that lowering the level of surface ambiguity would lead to a better conceptual understanding of the requirements and reduce the time needed for requirements analysis.

2.4.4 Feature extraction and selection

We developed two distinct feature extractor tools to extract feature values from SRS text. The features influence ambiguity at sentence level and discourse level respectively. We embedded Stanford's syntactic parser [Klein & Manning, 2003] to detect the presence of the quality characteristics influencing surface understanding (see Figure 2) in the text. Our feature extracted tools output feature vectors for each passage and sentence in our discourse level corpus and sentence-level corpus respectively. These features vectors were eventually used for building the discourse-level training data file and sentence level training data file. Section 5.4, 6.2.3 and 6.3.2 explain the entire process in details.

2.4.5 The Sentence Classifier

We then developed the sentence classifier using C4.5 decision tree learning algorithm [Quinlan, 1993]. The classifier has a training module, which can be trained using the sentence level training data and can dynamically generate a decision tree based on the data. This decision tree based classifier automatically classifies a sentence into ambiguous or unambiguous in terms of surface understanding. Chapter 5 explains the process of developing and training the classifier in details.

We experimented with the sentence classifier by training the classifier with different combinations of sentence level features and testing its performance, as compared to that of human annotation. The results showed high accuracy in performance, enough for the classifier to be applicable in the practical fields (see section 5.6 for details).

2.4.6 The Discourse Classifier

We then finally developed the discourse classifier, again using the C4.5 decision tree learning algorithm [Quinlan, 1993]. The classifier can classify a passage into ambiguous or unambiguous in terms of surface understanding. Chapter 6 explains the development and training process of the classifier in details.

We then experimented with the discourse classifier by training it with different discourse level features, and testing its performance against that of human annotation. The results showed high accuracy in performance affirming that the approach of using a text classifier is applicable in practical fields for detecting ambiguous passages in SRS documents. Sections 6.2.6, 6.3.4 and 6.4 present the above mentioned results and their detailed analysis.

2.5 Tools Used in the Thesis

Significant advancement in Natural Language Processing (NLP) technologies recently yielded many tools that are now being used in the attempts to detect ambiguity in SRS documents automatically and/or semi-automatically (see section 3.4, for a detailed survey on the currently studied tools). Our system also uses some of these tools in effort to detect ambiguity at the level of surface understanding. In this section, we will present a brief overview of these tools.

2.5.1 Dictionary

The term "Dictionary" in NLP denotes a repository of words, which are usually to be searched repeatedly for particular applications. The words can be indexed. A dictionary is either manually or automatically generated. Some recent work (e.g. [Fabbrini et al., 2001; Fantechi et al., 1994; Wilson et al., 1996]) use manually generated dictionary of ambiguous keywords to search for and. thus, detect ambiguity (chapter 3 discusses these studies). On the other hand, our system automatically generates a dictionary of ambiguous keywords using a heuristic (explained in chapter 5) and uses the dictionary later on in the detection process.

2.5.2 Syntactic Parser

A syntactic parser is another NLP tool that is being widely used in processing requirements specification documents, often in detecting ambiguities [Fabbrini et al., 2001; Gnesi et al., 2005; Osborne & MacNish, 1996]. In NLP, a parser refers to a tool that is able to chunk a group of words into different types of phrases, each with one grammatical role or function in a sentence. It, thus, relates a word of a sentence to all other words, usually by means of a tree structure, called the syntax tree. Some parsers can also output a directed acyclic graph that graphically shows the dependency relationship of words in a sentence. With this information acquired from a parser, the task of identifying words with ambiguous role in a sentence (at the level of surface understanding) becomes relatively doable. Our system uses the *Stanford Parser* [Klein & Manning, 2003] for the task of extracting ambiguous features (clues) from SRS documents. The parser has a built-in *POS tagger* and a *morphological stemmer* which are explained below:

A **Parts-of-speech (POS) tagger** is a tool used to find the parts-of-speech category of a word in the context of a sentence. Our system uses a variant of the most popular transformational POS-tagger, called the Brill tagger, developed by Eric Brill [1992], that comes with the *Stanford Parser*'s distribution.

A **stemmer**, on the other hand, is a tool that removes the inflections (suffixes and prefixes) from words to reduce them to their *stem*. Our system uses the *Stanford Parser*'s **morphological stemmer** that removes the inflections to the point where it retains the original POS category of the word. For example, "books" and "goes" are reduced to "book" and "go", but "badly" is not reduced to "bad" and remains unchanged, since it would then have changed its POS category in the sentence.

2.6 Text Classification

Text classification is a technique to classify text contents or documents into two or more categories based on different characteristics. It is being used for email classification and spam

detection [Kolcz & Alspector, 2001], clustering and organizing of documents [Larsen & Aone, 1999], in Information Extraction [Spertus, 1997] and Retrieval [Zhang & Varadarajan, 2006, Kleinberg, 1999] etc. A text classifier generally uses an implementation of a machine learning algorithm. In a supervised training method, values of discriminating features of the classes, which the classifier categorizes into, are collected from a large number of pre-classified documents. Then, with the aid of the machine learning algorithm, the classifier is 'trained' based on the feature values of the documents, which belong to all the different classes. Thus, when a new unclassified document is supplied to the classifier, it is able to classify the document into a class based on its feature values.

Although text classification technique has never been used in detecting ambiguity of SRS documents, we present here a case that uses text classification in a similar problem.

2.6.1 Case: Text Classification in Detecting Subjectivity

Wiebe *et al.* [2004] presented a comprehensive solution in detecting subjectivity of the natural language. They used features like frequencies of "Unique words" (words that appear only once in a discourse), "Adjectives", "Verbs" etc., and annotated sentences as either "subjective" or "objective". Then, they trained an implementation of the basic k-nearest neighbor algorithm [Dasarathy, 1991] using a cross-validation technique. On their tests, their system was able to classify 93% of the subjective sentences correctly.

2.7 Performance Measurement Metrics: Kappa

In this thesis, we evaluated performance mostly in terms of the degree of agreement between annotators, which is measured in **Kappa**. Cohen in 1960 introduced the Kappa index as a measure of agreement between raters, or "annotators" in our case. The Kappa index is denoted by κ, and refers to the following ratio:

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

Here, $P(A)$ is the proportion of total times that the annotators are observed to agree, and $P(E)$ is the proportion of the total times that the annotators are expected to agree.

Kappa has many desirable properties. For example, $\kappa = 1$ denotes complete agreement, and $\kappa > 0$ means the observed agreement among the annotators is higher than the expected chances of agreement. And, $\kappa \leq 0$ indicates that there is no agreement among the annotators beyond the chances of agreement among them.

There exist two versions of Kappa formulas to handle the situations of different number of annotators. One commonly used version deals with two annotators only (computes pair-wise agreement), while the other is used in the cases where the number of annotators is more than two. We will discuss both of these formulas in details for measuring the strength of agreement among the scores of our annotators in sections 4.4.2.2 and 4.4.2.3.

The interpretation of different values of Kappa indices differ with applications in different fields of study [Carletta, 1996; Kraemer et al., 2004; Krenn et al., 2004]. One most commonly used interpretation (put forth by Landis and Koch [1977]) is shown in Table 2:

Kappa κ Value	**Strength of agreement beyond chance**
< 0.00	Poor
0.01 - 0.20	Slight
0.21 - 0.40	Fair
0.41 - 0.60	Moderate
0.61 - 0.80	Substantial
0.81 - 1.00	Almost perfect

Table 2: Interpretation of the values of Kappa index [Landis & Koch, 1977]

2.8 Conclusion

Among all the tools and technique used today in the related fields, we find the above to best suit our approach. Detailed explanation on their usage in our work will be presented in the later chapters. The next chapter presents a brief overview on some of the most noted work in the field of requirements ambiguity detection.

Chapter 3

Literature Survey

3.1 Introduction

Different researches have addressed the problem of detecting ambiguities in natural language requirements specification. These studies typically use a small number of approaches, which are, although often similar in types of tools they use — radically different in the way they tried to detect ambiguities in SRS documents. In this chapter, we will review a few noteworthy studies by categorizing them according to their approaches.

3.2 Manual Detection Process

Manual detection is the most popular approach to detect and resolve the ambiguities of NL requirements specification. One of the early leading studies in this field was conducted by Bertrand Meyer [1985], showing the areas of a natural language requirements specification, where the specifier is more prone to make mistakes (see Table 1 in section 1.3.2). Meyer stressed on the point that natural language requirements specification are inherently ambiguous, and for resolving these ambiguities, use of formal specifications are absolutely necessary. However, for detecting such ambiguities, he explains the process of manually going though each word, phrase and sentence of the NL requirements specification text of his case study, and checking if they reflect any of the seven sins of the specifier.

Another study worth mentioning here is the one done by Kamsties *et al* [2001], who introduced five classes of different ambiguity problems of NL requirements specifications — each well-defined with practical examples, and used as items of a checklist for validating a requirements document. They are: Lexical Ambiguity, Systematic Ambiguity, Referential Ambiguity, Discourse Ambiguity and Domain Ambiguity. Table 3 describes these items briefly:

Item	Description
Lexical Ambiguity, Polysemy	Does a word in a requirement have several possibly related meanings? Be aware that *lexical ambiguity* arises in particular from the actual usage of a word in an RE context.
Systematic Polysemy	A systematic polysemy applies to a class of words: (1) The *object–class ambiguity* arises when a word in a requirement can refer either to a class of objects or to just a particular object of the same class. (2) The *process–product ambiguity* arises when a word can refer either to a process or to a product of the process. (3) The *volatile–persistent ambiguity* arises when a word refers to either a volatile or a persistent property of an object.
Referential Ambiguity	Can a phrase in a requirement refer to more than one object in other requirements? Check pronouns (it), definite noun phrases (the roads), and ellipses (... If not, ...).
Discourse Ambiguity	Does a requirement have several interpretations in relation to other requirements? This ambiguity arises when (1) words such as first, before, between, after, and last are used and can refer to several elements and when (2) adjectives, verbs, or noun phrases refer to more than one condition described before.
Domain Ambiguity	Is the requirement ambiguous with respect to what is known about the application, system, or development domain?

Table 3: Checklist for Ambiguity detection [Kamsties et al, 2001]

By describing the steps for ambiguity detection using this checklist, they argued in favor of manual inspection and stated that current NLP tools are not apt for proper disambiguation of NL requirements; rather, they are misleading. Their works also demonstrated dependence on formal specifications, e.g. UML models, especially for detecting domain ambiguities. Their suggested heuristics for detecting ambiguities involve attempting to develop UML models, and finding the points of contradiction and lack of information in the requirements specification. They recommended this process to be carried out by manual manipulation only. Their study concludes with the statement "one cannot expect to find all ambiguities in a requirements document with realistic resources" – even with such complete human involvement [Kamsties et al, 2001].

Manual detection is typically the most accurate approach; however, it is also the most expensive. Again, use of formalization is not well-understood by non-technical users as well. We also find Letier *et al* [2005] proposing the use of formal specifications to validate requirements.

3.3 Restricting Natural Language

Many other studies attempt to reduce the problems associated with unrestricted NL by limiting the scope of the language. Some use a new NL-like sublanguage, as in [Cyre, 1995; Lu et al., 1995], which is not truly NL. There are also others, who propose to restrict the grammar to consider only a subset of NL while writing a requirements specification [Denger et al., 2003; Fantechi et al., 1994; Rolland & Proix, 1992; Tjong et al. 2006]. Using a restricted language does simplify the task of detecting ambiguities, but imposes severe constraints on the requirements specifier's expression.

We will first look into the details of the study carried out by Heinrich *et al* [1999], where they proposed the use of a restricted language, called "Flexible Structured Coding Language (FSCL)", thoroughly defined by a fixed set of grammatical rules. The advantage about FSCL is that it has an unrestricted vocabulary, and it claims to be unambiguous enough to be translated into programming code automatically. Though the paper never defined the process of translation, the grammar it used has the potential to be unambiguous because of its strictness.

Fantechi *et al* [1994] suggested the use of a set of grammatical rules for aiding the translation from NL requirements specification to the formulae of "action-based temporal logic", called ACTL. They also have a domain-specific dictionary that helps the translation process. The grammar they defined can only deal with the possible structures of those NL sentences, which describes an expression of ACTL. This makes their grammar very limited for parsing a real requirements specification document. The ambiguities they could detect in their case study using this process were due to lack of information in the time and the quantification of an expression only.

A study conducted by Rolland and Proix [1992] was to translate natural language requirements specification to a form of semantic net, allowing a broad logical representation in conceptual schema. This required the use of a thoroughly defined dictionary that groups all kinds of verbs in six major categories: Agentive, Instrumental, Dative, Factitive, Locative and Objective. Each such category led to define a fixed set of grammar rules for parsing NL requirements statements into case notations. Their rules, thus, restrict the grammar of natural language used in specification. After translation, their system then follows a "paraphrasing process", using Chomsky's [1965] transformational grammar, to translate the conceptual schema of case notations back to NL-based statements. This allows the requirements elicitor to compare the natural language requirements specification given as input to the system, and with the one received as output from the system, and detect ambiguities. The work thoroughly relies on a fixed set of grammar rules and, although, claims to work with SRS written in NL, the case study they presented worked with an example SRS that contained sentences of a strictly simple structure, targeted to be caught by their fixed set of rules.

3.4 Using NLP Tools on Unrestricted Language

NLP techniques have advanced at tremendous speed during the past few years. And, for over a decade now, researchers in the fields of both NLP and software engineering, have been trying to merge NLP techniques with the tasks of requirements engineering. We know that requirements elicitation and validation is one of the key-phases of software's lifecycle that often takes considerably long time to finish with manual manipulation of information. A real-life requirements document can be lengthy and contain numerous words, phrases and sentences, where each of them becomes a candidate for possible ambiguities of different kinds. All these reasons made way for NLP techniques to come into the picture for tackling this problem. Researchers have introduced NLP in many different ways to detect ambiguity in requirements specification. The next sections present a brief survey on some of the most important research work in this area.

3.4.1 QuARS

Fabbrini *et al* [2001] and Gnesi *et al* [2005] addressed the issue by trying to measure the quality of a problem description, written in unrestricted NL. They initially made a survey on the contemporary studies revealing a number of defects that can exist in an NL requirements specification and listed those defects as "indicators" of poor-quality requirements specification. These are shown in Table 4 (extracted from [Gnesi et al., 2005]):

Characteristic	Indicators
Vagueness	*The occurrence of Vagueness-revealing wordings (as for example: clear, easy, strong, good, bad, useful, significant, adequate, recent, ...) is considered a vagueness Indicator*
Subjectivity	*The occurrence of Subjectivity-revealing wordings (as for example: similar, similarly, having in mind, take into account, as [adjective] as possible, ...) is considered a subjectivity Indicator*
Optionality	*The occurrence of Optionality-revealing words (as for example: possibly, eventually, if case, if possible, if appropriate, if needed, ...) is considered a optionality Indicator*
Implicity	*The occurrence of:* *- Subject or complements expressed by means of: Demonstrative adjective (this, these, that, those) or Pronouns (it, they...)or* *- Terms having the determiner expressed by a demonstrative adjective (this, these, that, those) or implicit adjective (as for example previous, next, following, last...) or preposition (as for example above, below...)* *Is considered an implicity Indicator*
Weakness	*The occurrence of Weak verbs is considered a weakness Indicator*
Under-specification	*The occurrence of words needing to be instantiated (for example: flow instead of data flow, control flow, .. , access instead of write access, remote access, authorized access, .. , testing instead of functional testing, structural testing, unit testing, .., etc.) is considered an under-specification Indicator.*
Multiplicity	*The occurrence of sentences having multiple subject or verb is considered a multiplicity Indicator*

Table 4: Quality Indicators of NL Requirements Specification [Gnesi et al., 2005]

Their studies proposed the use of their tool, called "*QuARS: Quality Analyzer for Requirements Specification*", for detecting sentences exhibiting different kinds of ambiguity in a problem description. Their tool first performs a lexical analysis over a problem

description using a POS tagger. It also syntactically parses the sentences using the MINIPAR parser [Lin, 1998], and finally, it combines both results for detecting the indicators of poor-quality requirement specification. It also contains an interface, called "View", for the requirements engineer to view the requirements statements by "clusters" having all the requirements regarding a specific function or property together. At every stage of processing, their tool requires the use of a different "modifiable" dictionary, which is specially created and modified for a particular stage of processing and for a specific problem domain by the requirements engineer. Their idea is thoroughly dependant on using a set of such special dictionaries, whose relevance and practical usage is uncertain. They developed their tool as a prototype for their idea, and it is said to produce a quality metrics of NL requirements specification. Again, their quality metrics are not well-defined to classify a problem description as ambiguous.

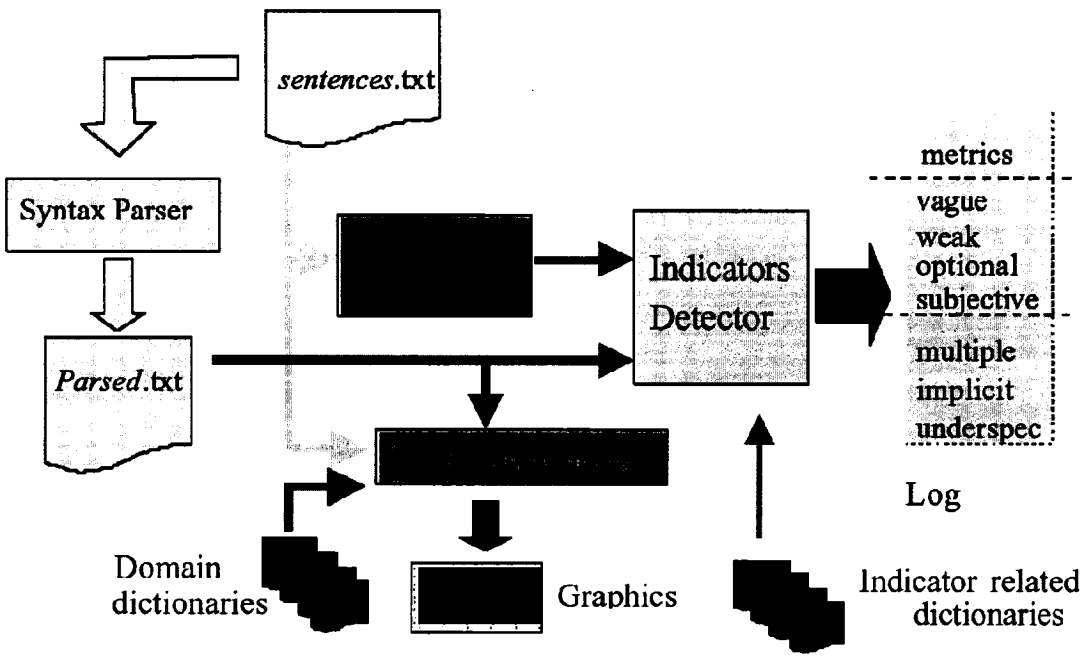

Figure 4: Concept of QuARS as proposed by Fabbrini *et al* [2001]

3.4.2 ARM

An automated tool for measuring the quality statistics of NL requirements documents, called "ARM: Automated Requirements Measurement", was developed by Software Assurance

Technology Center (SATC) of NASA. Its developers, [Wilson, 1996] and [Wilson et al., 1997], presented nine categories of quality indicators for requirements specification in detail. They are: **Imperatives**, **Continuances**, **Directives**, **Options**, **Weak Phrases**, **Size**, **Specification Depth**, **Readability** and **Text Structure**. The first five of these categories are based on frequencies of specific words occurring in ambiguity raising contexts. The remaining four are related to the organization of the entire requirements specification document. The results, derived from using the ARM tool, appeared to be more effective than others at detecting the level of ambiguity, but they ignored the use of sophisticated NLP methods, e.g. morphological analysis and syntactic analysis, which could have pointed out more ambiguity issues.

3.4.3 Newspeak

Osborne and MacNish [1996] used a parser to derive all possible parse trees of each sentences in a requirements specification document. Their system, called "Newspeak", then tries to detect ambiguity, if more than one parse tree exists for a particular sentence. Thus, their work only focuses on detecting ambiguous syntactic structure of sentences, completely ignoring the existence of different keywords that inherently induce ambiguity.

3.4.4 Circe

The work of Ambriola and Gervasi [1997] attempts to validate NL Specification with the aid of the user after deriving a conceptual model automatically from the requirements specification by their tool called Circe. Although their tool being funded by IBM is now available as a plug-in for Eclipse and is used in practical fields, it still does not consider the existence of ambiguities at the level of surface understanding, which can corrupt their conceptual model, making the errors tough for a user to detect from the model later on.

3.5 Conclusion

Our literature survey revealed that previous work in the area has attempted to flag ambiguous texts using various methods. However, evaluation of these (semi-) automatic methods are typically anecdotal or small-scale. To our knowledge, none has attempted to formally evaluate their results and compare them to human evaluations. Our study is an effort to evaluate the feasibility of such a task by analyzing how hard the task really is and how the automatic tools developed can compare to human performance. Our work on corpus annotation provides a benchmark for such an evaluation and an upper bound on what we can expect automatic tools to achieve.

Moreover, none of the current systems have attempted to use text classification techniques for detecting ambiguity in SRS documents. But, many studies have been conducted in other fields, e.g. Information Extraction [Spertus, 1997], Information Retrieval [Zhang & Varadarajan, 2006], Question Answering [Wiebe et al., 2003], Text Classification [Wiebe et al., 2004] etc. that detects the subjective elements in language using text classification or data mining techniques. We, therefore, believe that with proper training, a text classification system can prove to be of immense benefit in a similar problem, i.e. to detect ambiguity in software requirements specification.

After we have explained the process of developing our text classification system that detects ambiguity in SRS documents, we will compare the performance of all the tools mentioned in this chapter against that of our system in terms of various topics (see Table 22 of section 6.5). The next chapter explains the manual process of building our annotated corpus and the evaluation of human performance in terms of inter-annotator agreement.

Chapter 4

Corpus Annotation

4.1 Introduction

To build our classifier we needed an annotated corpus for the purpose of training and testing. Since there exists no standard corpus on SRS documents that we could use for our task, we had to build our own sets of annotated corpora.

Our idea was to select a number of human annotators, provide them with a detailed guideline of annotation, and ask them to annotate segments of NL requirements specifications as "Ambiguous" or "Unambiguous". The resulting data would be used as our corpora.

We randomly chose 25 problem descriptions from *ACM's OOPSLA DesignFest®* online source (http://designfest.acm.org/) for the task of building our corpora. Our work initially focused on *Discourse Annotation*, which means that the annotators were asked to annotate a discourse or a *passage*, in our case, (see section 4.3.1, for the definition of *passage* used in this thesis) from a problem description as "Ambiguous" or "Unambiguous" in terms of Surface and Conceptual Understanding separately. The annotations revealed the complexities of detecting ambiguity at the level of Conceptual Understanding without the means of conceptual modeling, which eventually made us put the work out of the scope of this project. Thus, the task of discourse annotation at the level of Surface Understanding formed our *Discourse-level Corpus*, where we had two sets of "Ambiguous" and "Unambiguous" passages.

One of our annotators performed *Sentence Annotation* prior to her task of discourse-level annotation. She annotated 1211 sentences, all from our 25 problem descriptions, as "Ambiguous" or "Unambiguous" in terms of Surface Understanding. Her work was later on reviewed and refined by two annotators following our Annotation Guideline. The result was used to compose our *Sentence-level Corpus*, containing sets of both "Ambiguous" and "Unambiguous" sentences.

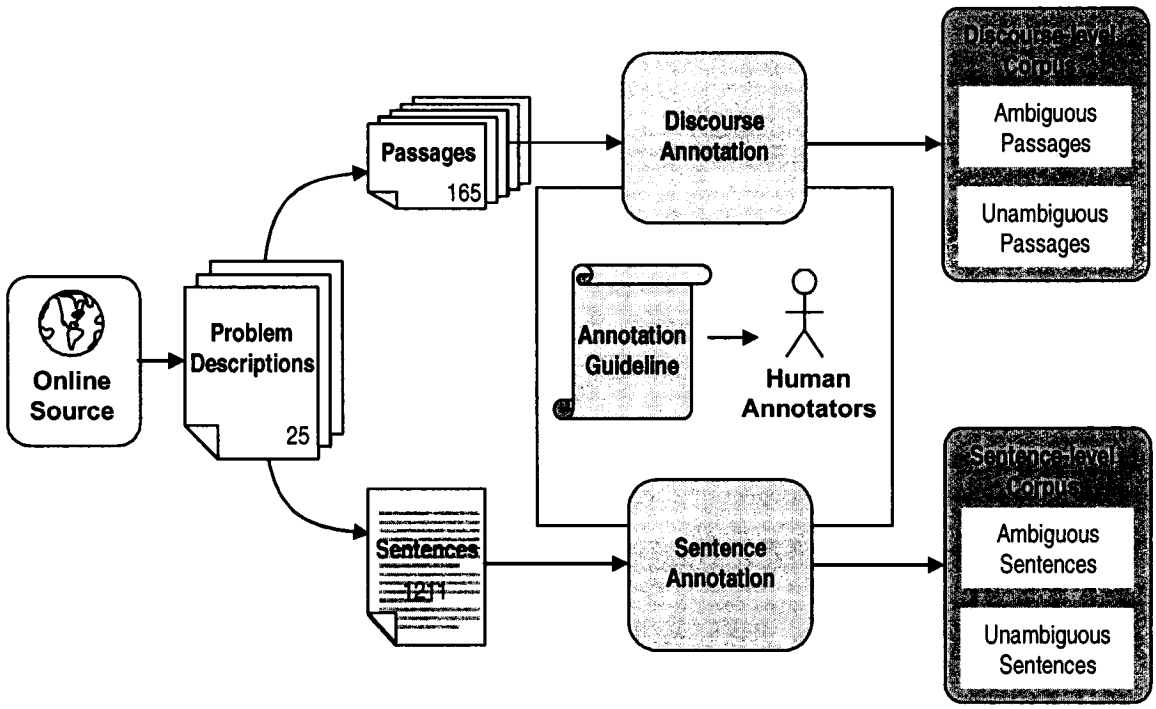

Figure 5: Corpus Annotation

Figure 5 illustrates the steps of our corpus annotation process. The following sections explain the process in details.

4.2 The Annotators

We had four persons working on the texts as annotators (*Annotator1*, *Annotator2*, *Annotator3* and *Annotator4*) — all with software engineering backgrounds, but from different fields of computer science. There were two professors — Dr. Kosseim and Dr. Ormandjieva, and two

graduate students, Mr. Hussain and Ms. Seresht, all from the Computer Science and Software Engineering department of Concordia University.

Among our annotators, Dr. Ormandjieva (*Annotator3*), being a professor in Software Requirements Specification, was the only expert we could avail that held profound experience in Requirements Analysis. The others had working experience in the fields of software engineering and were familiar with the concepts of Requirements Analysis.

4.3 The Corpora

We collected 25 problem descriptions related to software requirements specification through a random selection process from *OOPSLA DesignFest*®, an online source (url: http://designfest.acm.org/). Annotating an entire problem description of several pages, would be difficult and not particularly informative if we later wish to identify specific features for an automatic classification task. We therefore chose to split up our collection of problem descriptions into smaller segments that derived two different sets of corpora: the *Discourse-level Corpus* and the *Sentence-level Corpus*. They are explained in the following sections:

4.3.1 Discourse-level Corpus

For our Discourse-level Corpus, we extracted 165 *passages* from our collection of 25 problem descriptions, where each *passage* contained 140 words on average. Our definition of a *Passage* in this thesis was as follows:

Passage: *A passage is the smallest but stand-alone section of a problem description. It typically consists of one or more paragraphs under a section heading. Such a unit is selected so that the complete description of the concepts that it introduces remains within its scope.*

While extracting the passages, we intended for them to be trimmed down to a smallest unit of discourse that were understandable without additional context. We, thus, introduced the following minimal modifications:

1. Any section headings associated with the selected passage was removed.

2. Images and Tables within the passage were replaced with the tags "(IMAGE)" and "(TABLE)" respectively, to eliminate noise.

4.3.2 Sentence-level Corpus

Our *sentence-level corpus* consisted of 1211 sentences, all extracted from our collection of 25 problem descriptions. The sentence boundaries (".", "?" and "!") were manually detected for the process of extraction. Each of the extracted sentences contained 19 words on average.

4.4 Discourse Annotation

4.4.1 Training and Annotation

4.4.1.1 Pilot Study

We conducted a pilot study before starting our actual annotation phase to check if our annotators correctly understood the task. The study was carried out on 20 samples, which were randomly selected from the 165 passages of the *discourse-level corpus*.

We asked our annotators to classify the samples into one of the two classes: "Ambiguous" and "Unambiguous" and do this for both Surface and Conceptual Understanding separately. However, no specific guidelines were initially given to them, apart from an informal definition of surface versus conceptual understanding (as presented in sections 1.4 and 1.5).

After evaluating their results, we found significant disagreements among the annotators in the classifications. By interviewing the annotators, we realized that an intuitive definition of surface and conceptual understanding was not enough. Everyone seemed to have their own views on what was ambiguous and what was not. Some annotators were more lenient, while others were strict. This indicated the need of a singular annotation guideline that is thoroughly defined with examples, and that all the annotators should follow.

Interviewing annotators also revealed their discomfort in annotating a discourse with a binary decision, e.g. "Ambiguous" or "Unambiguous", for a particular level of understanding. They all unanimously agreed that they would be more comfortable in grading the passages on a scale of zero to ten for both Surface and Conceptual Understanding separately. Their grades

would later be translated into "Ambiguous" or "Unambiguous" by setting a threshold on the scores.

Thus, the pilot study led us to conclude that the task of annotation should be done based on a strictly defined annotation guideline, and should also provide the convenience for the annotators to make fuzzy decisions in classifying the passages with different levels of understanding.

4.4.1.2 Annotation Guidelines

In light of the pilot studies, we, therefore, developed more specific guidelines and clearer examples of what were to be considered ambiguous in terms of surface and conceptual understanding separately. The annotators were to score all passages of our corpus (*the higher their scores for a passage, the less ambiguous it would be*), and the annotation guidelines indicated what to look for in a passage, but, did not give any strict instructions on what score to be selected, to allow the annotators to have their freedom in scoring. We developed these guidelines based on our quality model [Ormandjieva et al., 2007] considering previous work in the field, such as [Fabbrini et al., 2001; Kamsties et al., 2001; Gnesi et al., 2005; Meyer, 1985; Wilson, 1997; Wilson et al., 1996]. The guidelines are as follows:

A. Scoring Guidelines for Surface Understanding

1. **Long sentences are *bad*:**

   Following the study of [Wilson, 1997], we identify long sentences as a negative quality of SRS documents. Thus, while grading the level of surface understanding of a passage, if the density of long sentences were found to be higher than acceptable limit (which may be set differently by different annotators according to their preferences), we gave a considerably lower score, increasing the chance of a passage to be annotated as "Ambiguous" in terms of Surface Understanding.

   The characteristics of long sentences are described as follows:

- One sentence containing more than 256[1] characters (including white-space character), or more than two lines of text can be considered a long sentence.

- One sentence with more than two main (principle) verbs can be considered a long sentence. An example of such a long sentence (with the main verbs underlined) is as follows:

  > *We would like you to pay most of your attention to the fact that there are so many different cells in the network that need to be managed in a common way, but are still of very different types.* [Kriens, 1996]

- One sentence containing more than three commas (or semi-colons) can be considered a long sentence. An example of such long sentence is as follows:

  > *The system assigns the call to a collector based on a user-defined algorithm based on the stage of delinquency the borrower is in, the workgroup to which a collector belongs, the specific collector that placed the last call, the result of the last call, and the relative availability of collectors to service calls.* [Best, 1995]

- One sentence containing a main verb (i.e. a clause) inside brackets is considered a long sentence. An example of such a long sentence (with the main verbs underlined) is as follows:

  > *The Rumbling Range National Forest (RRNF) buys two new arrays of sensors (Sensor arrays are so named because they are collocated and allow the central site to glean more information than just the sum of that from each individual sensor).* [Heliotis et al., 2003]

2. Too many adjectives and adverbs are *bad*:

A passage containing too many adjectives, e.g. *clear*, *well*, *easy*, *efficient*, *adequate*, *strong*, *weak*, *good*, *bad*, *high*, *low*, *fast*, *slow* etc., and adverbs could be given a lower score to annotate it as "Ambiguous" in surface understanding. Work of Fabbrini *et al*

[1] The number of characters was chosen arbitrarily

[2001], Gnesi *et al* [2005] and Wilson *et al* [1996] also uses adjectives and adverbs as negative quality characteristic of SRS documents. Thus, if the density of adjectives and/or adverbs is higher than the acceptable limit (set by an annotator according to his/her preference), the annotator should reduce his/her passage score.

3. **Too many verbs in passive voice are *bad*:**

Similarly to [Gnesi et al., 2005] and [Wilson 1997], we consider that *Verbs* in passive voice is a negative quality characteristic of SRS documents. A passage containing too many sentences in passive voice should be given a lower score for it to be annotated as "Ambiguous" in terms of surface understanding. An example of such sentence is as follows:

> *Most commonly, the orders are matched in price/time priority.*
>
> [Layda, 2000]

— Here, it is unknown who or what matches the order.

4. **Directives are *Good*:**

If a directive word or phrase was found in a passage, e.g. "for example", "note:", "e.g.", "such as", "(IMAGE)", "(TABLE)" etc., it should regarded as a positive quality characteristic of an SRS document. Thus, the annotators may decide to increase his/her scores for that passage relieving the chance for it to be annotated as "Ambiguous" in terms of Surface Understanding because of a higher score.

B. Scoring Guidelines for Conceptual Understanding

Meyer's "Seven Sins" are *bad*:

Using Meyer's [1985] definitions of the "Seven Sins of Specifier", described in section 1.3.2, Annotators were to search for the "seven sins" in the passages (*Noise*, *Silence*, *Ambiguity*, *Over-specification*, *Contradiction*, *Forward Reference* and *Wishful Thinking*). While grading the level of Conceptual Understanding of a passage, an annotator should lower his/her score upon finding the mistakes, like "Noise",

"Silence", "Ambiguity", "Over-specification", "Contradiction", "Forward Reference" and "Wishful Thinking" (see section 1.3.2, for details).

4.4.1.3 The Standard Interpretation of Scores

We allowed our annotators to score the passages on a grading scale of (from zero to ten). This method was undertaken to address the issue of the annotators' discomfort in annotating a passage with a binary decisions: "Ambiguous" or "Unambiguous". The numerical scores were to be converted into a binary decision later. To allow for a better classification, we limited the annotators by giving them a standard interpretation of the scale:

- *An Ambiguous passage should be given a score in the range* **[0,5)**.
- *A passage that could be either way should be given a score exactly equal to* **5**.
- *An Unambiguous passage should be given a score in the range* **(5,10]**.

Annotators were allowed to add their own interpretations of these scores — for example, scores $\geqslant$7.5 and $\leqslant$10 could mean "very clear", meaning the description in the text was almost perfect, or score $\geqslant$0 and <3 could mean "very ambiguous". These numbers and interpretations for "very clear", "not so clear" etc. were based an annotator's own choice, and for their own comfort in scoring. Thus, each annotator was allowed to have their own interpretation as long as the standard interpretation of "Ambiguous" and "Unambiguous" were respected.

4.4.1.4 Task of Annotation

The annotators took the 165 passages of our *discourse-level corpus* and independently ranked them with scores from 0 to 10, considering surface and conceptual understanding separately.

The annotators gave about 5.549 hours of effort on average in scoring the passages for both surface and conceptual understanding. The total times each annotator spent performing these annotations are presented in Figure 6:

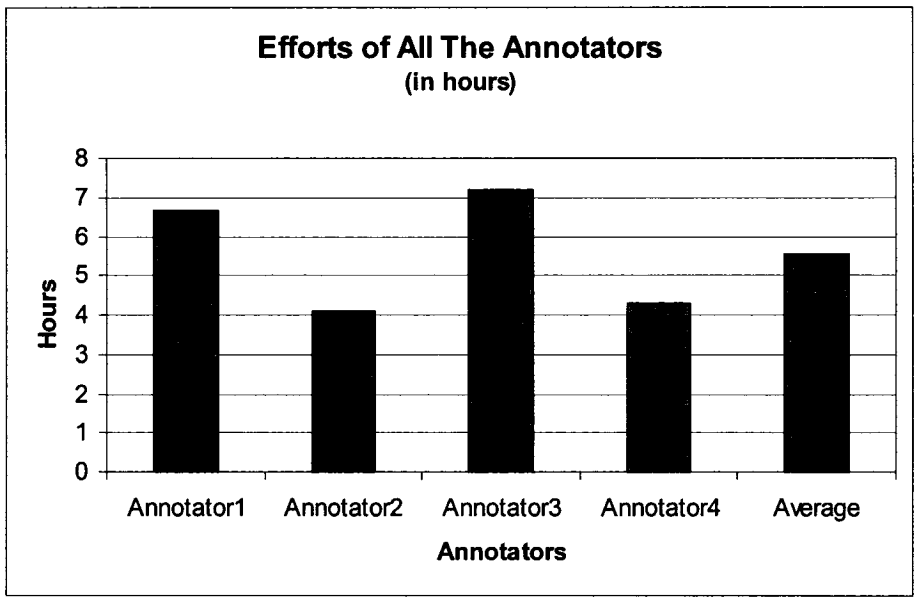

Figure 6: Efforts of All the Annotators (in hours)

On average, it took 2.018 minutes to annotate each passage. Considering that each passage has, on average, 140 words, we could say that the annotators read at a speed of 69 wpm (words per minute). It shows that the task of detecting ambiguity was a difficult one, considering the average reading speed of a "normal reader", which is around 240 wpm [Just & Carpenter, 1987].

4.4.2 Results and Analysis

4.4.2.1 The Gold Standard

For each instance (passage) of our discourse-level corpus, we translated its scores given by the annotators' to binary annotations by first taking the median of its scores, and then, translating the numeric value of the median as follows:

- If the median is less than 5 (five),
  - The gold standard of the passage is interpreted as "Ambiguous".
- If the median is exactly equal to 5 (five),
  - The gold standard of the passage is interpreted as "Either way".
- If the median is greater than 5 (five),
  - The gold standard of the passage is interpreted as "Unambiguous".

According to the Median Voter Model [Congleton, 2004], the gold standard, evaluated in this way, reflects the decision of the majority. Let us consider the following passage for an example:

> *Case Management is a business function common to government health benefit programs, and insurance and financial organizations. The problem you are here to solve today is to develop an object oriented framework for use in developing specific case management systems to support case workers in these industries. These systems must provide access to all necessary information for case workers to process work or respond to customer service needs. They may also provide workflow and business rules processing to support case workers. As the main incoming channel for cases is frequently the telephone, the system must have sub-second response times and be intuitive to use.* [Layda, 2000]

The annotators' actual scores for this passage are shown in Table 5.

	Annotator1	Annotator2	Annotator3	Annotator4
Score for Surface Understanding	8	4	6	8
Score for Conceptual Understanding	2	2	5	3

Table 5: Example of the annotators' scores on one passage

As we calculate the median of the scores and translate its value according to the aforesaid rule, we get the gold-standards shown in Table 6.

	Annotator1	Annotator2	Annotator3	Annotator4	Median	Gold Standard
Surface Understanding	8	4	6	8	7	**Unambiguous**
Conceptual Understanding	2	2	5	3	2.5	**Ambiguous**

Table 6: Example of the annotators' scores on the same passage with their median and the Gold Standard

Thus, for Surface understanding, the resultant distribution of our discourse-level corpus is shown in Figure 7.

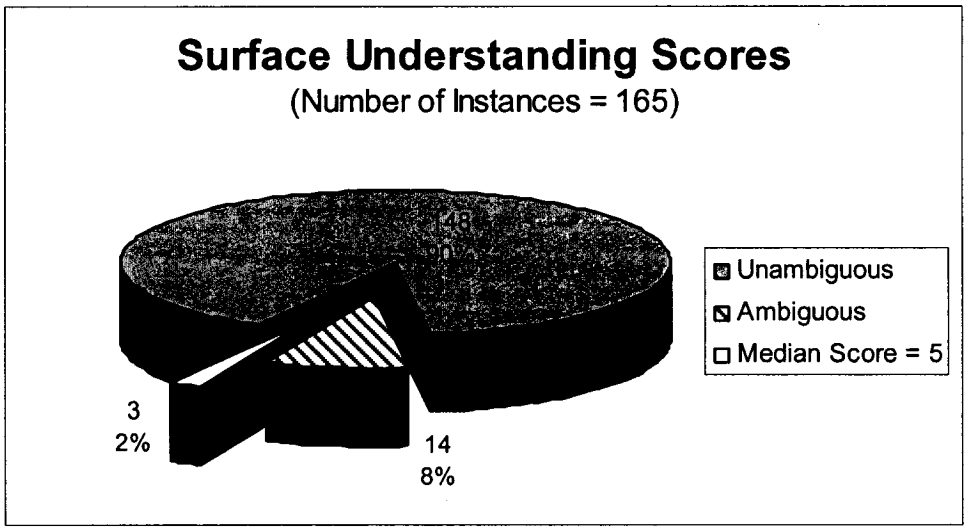

Figure 7: Distribution of the *discourse-level corpus* based on the scores of surface understanding

Figure 7 shows that of 165 passages from our *discourse-level corpus*, 148 (90%) were annotated as "Unambiguous" in terms of surface understanding. 14 (8%) of them were "Ambiguous" and 3 (2%) of them were the ones with median score 5 (i.e. they can go either way).

Again, for conceptual understanding, the distribution of the corpus is shown in Figure 8.

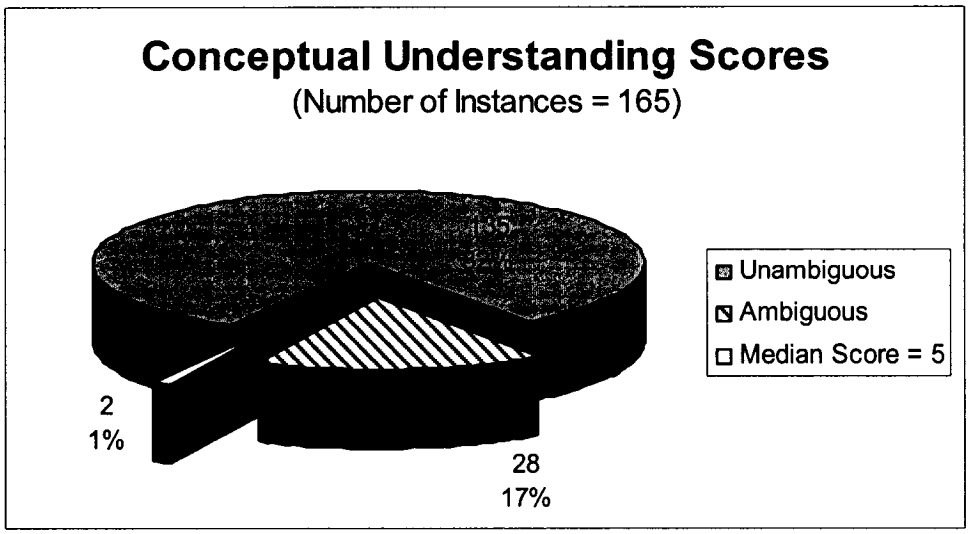

Figure 8: Distribution of the *discourse-level corpus* based on the scores of conceptual understanding

Figure 8 shows that 135 (82%) of 165 passages were annotated as "Unambiguous" in terms of Conceptual Understanding. 28 (17%) of them were "Ambiguous" and 2 (1%) of them were the ones with median score 5.

4.4.2.2 Pair-wise Agreement with Gold Standard

The first type of agreement measure that we computed is pair-wise agreement with the gold-standard. For each annotator, we computed Cohen's [1960] *Kappa index* (see section 2.7 for details on *Kappa*) with the gold standard. To calculate this Kappa we used six measures (four of them are listed in Table 7) as follows:

- P_{11}: the proportion of times both the annotator and the gold-standard marked a passage as "Unambiguous".
- P_{22}: the proportion of times both the annotator and the gold-standard marked a passage as "Ambiguous".
- P_{21}: the proportion of times the annotator marked a passage as "Ambiguous", but the gold standard marked it as "Unambiguous".

- P_{12}: the proportion of times the annotator marked a passage as "Unambiguous", but the gold standard marked it as "Ambiguous".

- $P(A)$: the proportion of times the annotator agrees with the gold-standard, computed as

$$P(A) = P_{11} + P_{22}$$

- $P(E)$: the proportion of times the annotator is expected to agree with the gold standard by chance. If we do not assume random distribution, then

$$P(E) = ((P_{11} + P_{21}) \times (P_{11} + P_{12})) + ((P_{22} + P_{21}) \times (P_{22} + P_{12}))$$

		Gold-standard	
		Unambiguous	**Ambiguous**
Annotator	**Unambiguous**	P_{11}	P_{12}
	Ambiguous	P_{21}	P_{22}

Table 7: Contingency table to compute the Kappa index

The Kappa index for the i^{th} annotator was then computed as specified in [Cohen, 1960]:

$$\kappa_i = \frac{P(A) - P(E)}{1 - P(E)}$$

We calculated Kappa for each of the annotators and for each level of understanding separately. For each level of understanding, we also computed the average Kappa as follows:

$$\kappa_{avg} = \frac{1}{n} \cdot \sum_{i=1}^{n} \kappa_i$$

— where n is the total number of annotators.

The resulting values of Kappa indices are shown in Figure 9.

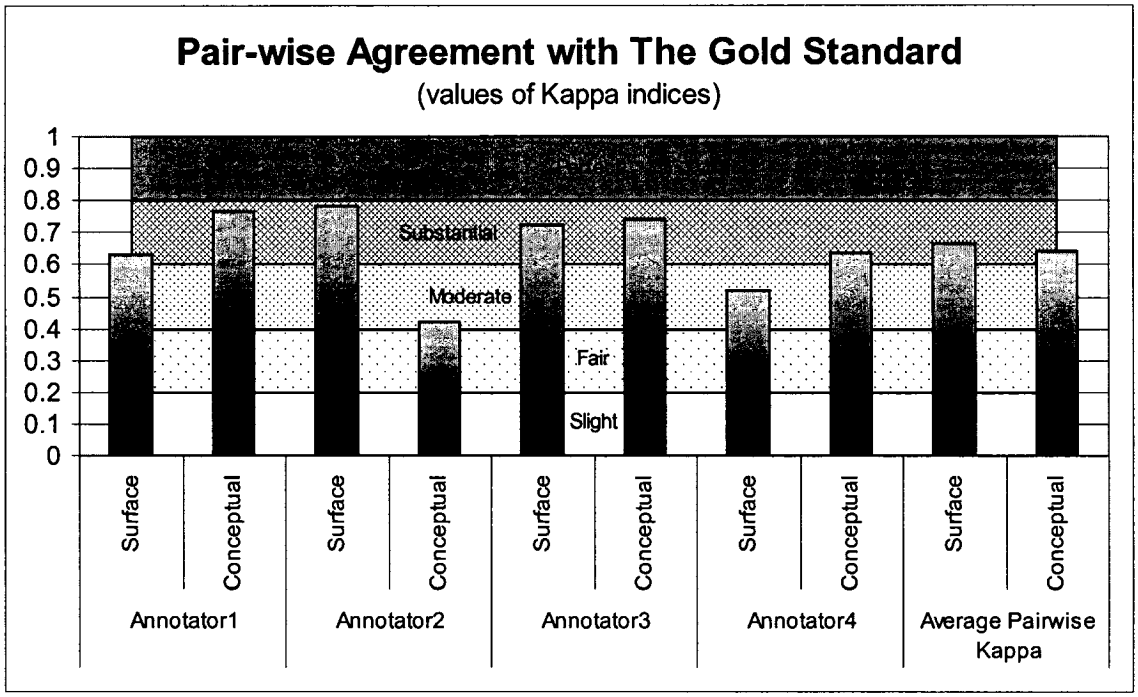

Figure 9: Pair-wise inter-annotator agreement with gold-standard

Figure 9 shows the results of this analysis, revealing how strongly each annotator agrees with the gold-standard, which is the decision of the majority of the annotators (according to the Median Voter Model [Congleton, 2004]). Therefore, here, the average of the pair-wise Kappa, simply denotes the strength of the gold-standard itself. According to the interpretation of Kappa values given by Landis and Koch [1977] (see Table 2 of section 2.7), on average the annotators agreed with the gold-standard at a "Substantial" level for both Surface and Conceptual Understanding. Only Annotators2 and Annotators4, however, have Kappa values at "Moderate" level of agreement.

4.4.2.3 Multiple Annotator Agreement

Multiple-annotator agreement was also used to find the value for a modified Kappa index, which is introduced in [Fliess, 1971] to find a combined Kappa value for a variable number

of annotators. The formulas here can deal with different number of annotators (greater than two) annotating different samples for a particular level of understanding.

If the total number of samples is equal to n, the number of annotations for the i^{th} sample is m_i, and the number of "Unambiguous" annotations for the i^{th} sample is x_i, then we calculate the mean number of annotations per sample, $\overline{m}$, as follows:

$$\overline{m} = \frac{\sum_{i=1}^{n} m_i}{n}$$

— and the overall proportion of "Unambiguous" annotations, as follows:

$$\overline{p} = \frac{\sum_{i=1}^{n} x_i}{n.\overline{m}}$$

Therefore, the combined Kappa according to Fleiss [1971] is as follows:

$$\kappa_C = 1 - \frac{\sum_{i=1}^{n} \frac{x_i(m_i - x_i)}{m_i}}{n.(\overline{m}-1).\overline{p}.(1-\overline{p})}$$

The results of our annotations in terms of the combined Kappa measure are presented in Figure 10.

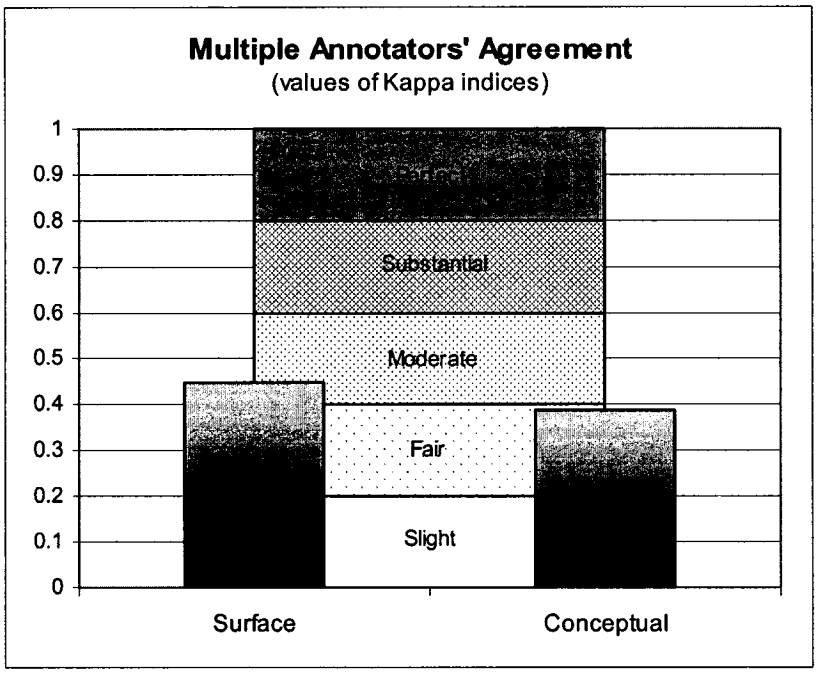

Figure 10: Multiple Annotators' Agreement

Figure 10 shows the multiple annotators' agreement, depicting how strongly the annotators altogether agreed to their decisions. While pair-wise agreement with the gold-standard showed "Substantial" level of agreement of the annotators with the gold-standard, here, however, the combined Kappa falls short, and is marginally "Moderate", according to Landis and Koch [1977], which, nevertheless, makes all the results good enough for automating the task of classifying a discourse in terms of ambiguity.

This concluded our work of building the discourse-level corpus.

4.5 Sentence Annotation

4.5.1 For Surface Understanding Only

Our primary intention was to detect ambiguity in SRS documents, and thus, at the discourse level. So, we started off our annotation work at the discourse-level first. But, examining the *Annotation Guideline* (described in section 4.4.1.2) for discourse annotation, we found that

all the characteristic, which were instructed to be looked for by the guideline for detecting ambiguity at surface understanding, had their scope of influence limited to the sentence they belong. On the other hand, according to the guideline, the characteristics that influence conceptual understanding, i.e. Meyer's [1985] "Seven Sins", can only exist within the context of a discourse.

We, therefore, considered sentence annotation to have an annotated sentence-level corpus for the purpose of building a sentence classifier that can classify sentences based on ambiguity in surface understanding. This could eventually aid the ambiguity detection process at discourse-level for surface understanding. Related work of [Fabbrini et al., 2001; Kamsties et al., 2001; Wilson, 1997] also attempts to detect ambiguity in SRS, but at the sentence level.

4.5.2 Informal Task of Annotation

Due to limitation of time and resources, we could only have one annotator for our task of sentence annotation (see chapter 8 for details about the limitations of this project). *Annotator3*, being a professor of Software Requirements Specification and the only expert in Requirements Analysis, performed the task of Sentence Annotation, based on the same Annotation Guidelines, that was defined to annotate passages in terms of Surface Understanding (see section 4.4.1.2). Using for the same characteristics of ambiguity in context of a sentence, instead of a passage, we found the following phenomena that made a sentence ambiguous in most cases:

- Long sentence: A long sentence can be identified by many commas or semi-colons, several main verbs or clauses in a sentence, the use of parenthesis etc.
- Adjectives and adverbs in a sentence.
- Verbs in passive form.

Thus, our annotator, *Annotator3*, annotated the 1211 sentences. The sentences contained 19 words on an average. Her annotations were later reviewed and refined by both *Annotator1* and *Annotator3*, on repeated sessions to improve the quality of the annotation of the Sentence-level corpus.

4.5.3 Results

The resultant distribution of the sentence-level corpus is shown in Figure 11:

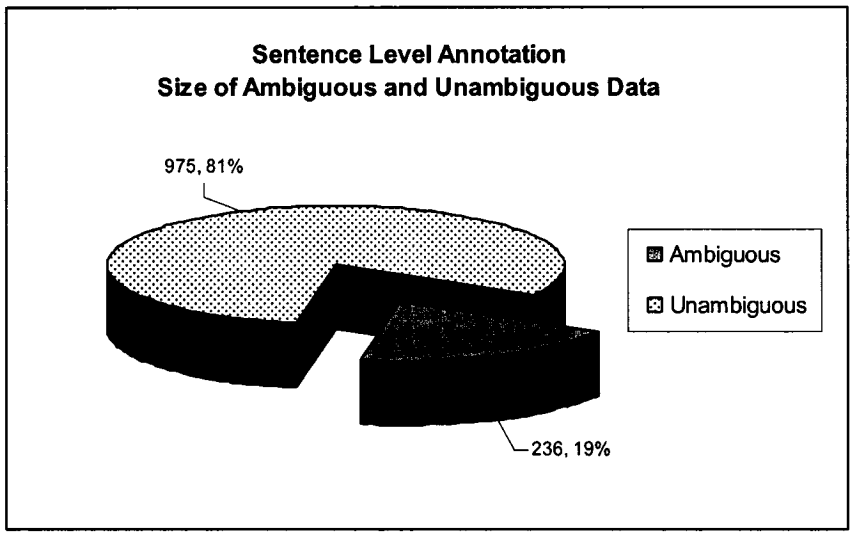

Figure 11: Distribution of Sentence-level Corpus after Sentence Annotation took place

There were in total 1211 sentences, 975 (81%) of them are "Unambiguous", and 236 (19%) of them are "Ambiguous". Ambiguous sentences contain 21 words on an average, whereas Unambiguous sentences contain 18.56 words on an average.

This concluded our task of building the sentence-level corpus.

4.6 Conclusion

Our work on discourse-level annotation establishes the fact that automation of detecting ambiguity is plausible at the level of both surface and conceptual understanding. Now, we know that there exists a lack of NLP tools efficient enough to perform deeper semantic analysis required for detecting ambiguity at the level of conceptual understanding. Our experiments on discourse annotation data, therefore, were not really directed to show the feasibility of using a classifier. Rather, the level of the annotators' agreement ascertains an important notion that the eventual extraction of right discriminating features from the text will lead to a successful classification of ambiguity in both surface and conceptual

understanding. But, this would require more improvement in NLP tools in terms of accuracy at the semantic level, especially for the case of detecting ambiguity in conceptual understanding.

Our work on sentence annotation, although, followed an informal process of ad-hoc corpus annotation, provided the necessary corpus for training and testing our sentence classifier. The sentence classifier, explained in the next chapter, was used later on in building the discourse classifier, and proved to be an important addition to our work.

Chapter 5

Sentence-level Classification

5.1 Methodology

Our objective here was to build a text classification system that could classify sentences as "Ambiguous" or "Unambiguous" in terms of surface understanding. Although our ultimate target was to build a classifier that can classify a discourse in terms of its ambiguity, we focused on building a similar classifier at the sentence-level to assess the achievability of automating the task of ambiguity detection at the more limited scope of the sentence. We used our *sentence-level corpus* (described in section 4.5 of the previous chapter) for training and testing our classifier. Each instance of this corpus was a sentence, which had been pre-annotated as "Ambiguous" or "Unambiguous" by *Annotator3* (reviewed and refined by *Annotator1* and *Annotator3*).

We used the *Stanford Parser* [Klein & Manning, 2003] (equipped with *Brill's POS tagger* [Brill, 1992] and a morphological stemmer) to extract a list of syntactic features from each of the training instances (sentences) of the sentence-level corpus. We identified these features as candidates that might have some influence in inducing ambiguity to a sentence in terms of surface understanding. The features valid for detecting ambiguity were then automatically selected based on their ranks in *Likelihood Ratios* (which will be explained in section 5.4) in the sentence-level corpus. We also used the corpus to generate lists of ambiguous keywords dynamically, by comparing the *Likelihood Ratios* of the keywords of particular POS categories (see section 5.4 for more details). The process of development will be explained in

details in section 5.4, where we present our *Features Extractor* tool. Figure 12 illustrates the workflow of the process.

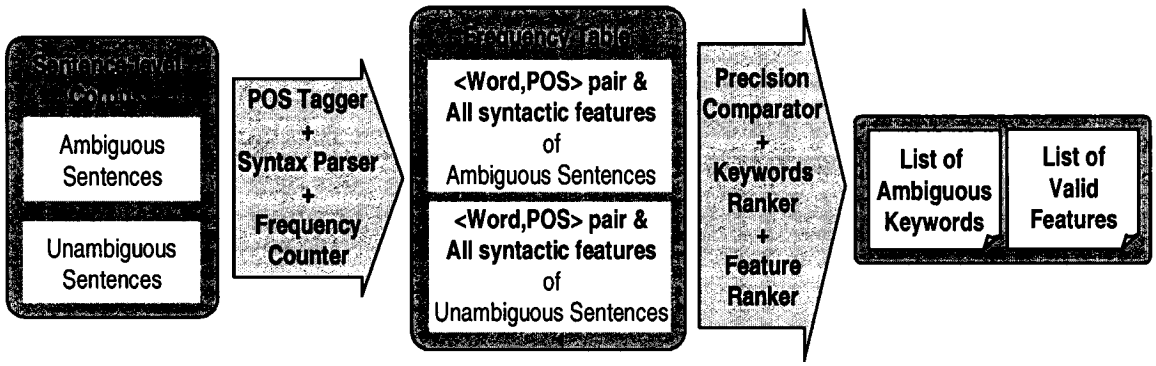

Figure 12: Automatic Feature Validation and Dynamic Generation of Ambiguous Keywords

Our sentence-level *Feature Extractor* tool extracts features required for detecting ambiguity in a sentence. It counts the frequency of occurrences of valid features and ambiguous keywords only in each of the sentences of our corpus and stores them in the training data file. The training data file also holds the corresponding annotation of each sentence assigned by *Annotator3*. Our sentence classifier was trained using this file. Thus, the classifier then can detect ambiguity by classifying the unclassified sentences into two categories: "Ambiguous" and "Unambiguous" sentences. Figure 13 illustrates the steps of the classification.

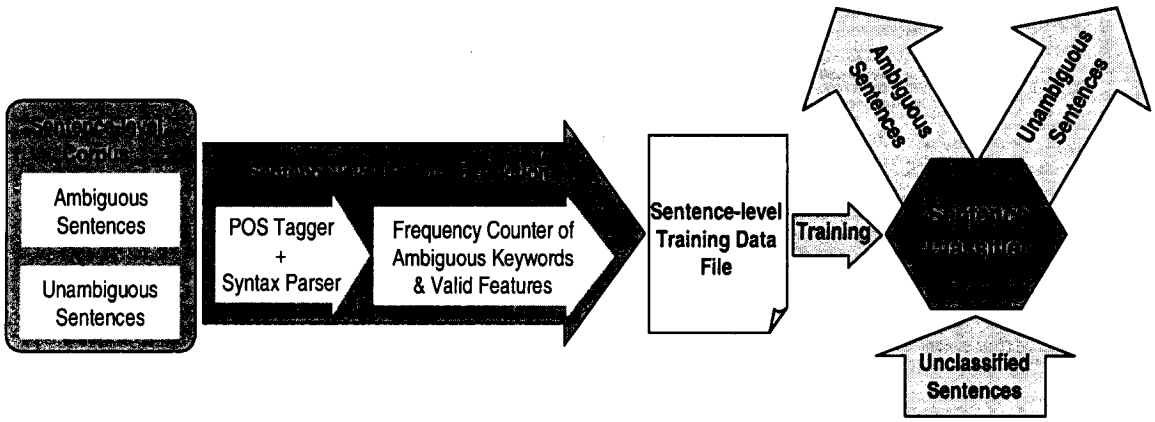

Figure 13: Steps of Sentence-level Classification

The following sections explain the process of sentence-level classification in details, and discuss its evaluation through an experiment.

5.2 Preprocessing

We first divided our sentence-level corpus into two separate sets: (1) Ambiguous Sentences (*CorpusA*), containing all the sentences that were annotated as "Ambiguous" in terms of surface understanding, and (2) Unambiguous Sentences (*CorpusU*), containing the rest of the sentences which are annotated as "Unambiguous".

As we have seen earlier in section 4.5.3, there were in total 1211 sentences in our corpus, 975 (81%) of them were annotated as "Unambiguous", while 236 (19%) of them as "Ambiguous". Therefore, the sizes of *CorpusU* and *CorpusA* are 975 and 236, respectively.

We used *CorpusU* and *CorpusA* as they were for feature extraction. But, to train our classification algorithm, we needed our corpus to be of nearly equal size. So, during the creation of the training data file, we reduced the size of *CorpusU*, by taking only the first 236 sentences from it; thus, making its size equal to *CorpusA*.

5.3 List of Features

We tried to choose a list of lexical and syntactic features, which were extractable by the a syntactic parser, e.g. *Stanford Parser* [Klein & Manning, 2003] and might have an influence in making a sentence ambiguous in terms of Surface Understanding. The features are as follows:

1. *Number of Words in a sentence*
2. *Number of Adjectives* in a sentence*
3. *Number of Adverbs* in a sentence*
4. *Number of Passive Verbs* in a sentence*
5. *Number of Parentheses* in a sentence*
6. *Total Number of Tokens* inside (or, between) all pairs of parentheses in a sentence*

7. *Total Number of all forms of Verbs* inside (or, between) all pairs of parentheses in a sentence*
8. *Number of Punctuations* in a sentence*
9. *Number of Conjunctions* in a sentence*
10. *If a sentence is a Fragment* (incomplete sentence)*

The frequency of the features above, marked with asterisk (*), are *normalized* (i.e. arithmetically divided) by the total number of words in a passage.

From this list, features 1, 2, 3, 4 and 10 are also used by [Fabbrini et al., 2001; Kamsties et al., 2001; Gnesi et al., 2005; Wilson, 1997]; and, the ambiguity characteristics of features 1, 2, 3, 4, 6, 7 and 10 are also mentioned in our annotation guideline, described in section 4.4.1.2.

5.4 Feature Extraction at the Sentence-level

We developed a sentence-level *Feature Extractor* tool written in Java to extract the values of the features likely to make a sentence "Ambiguous" or "Unambiguous" in terms of surface understanding. This program performed 4 different tasks, which are described below:

1. **Frequency Generation of Parser Features:**

   The Feature Extractor tool extracts sentences from both *CorpusA* and *CorpusU*, and feeds them one-by-one to the *Stanford Parser* [Klein & Manning, 2003] for POS tagging and syntax parsing. The values of the features mentioned in section 5.3 are then counted for each sentence of the two corpora. In addition, the *Feature Extractor* tool also counts the total of number of occurrences of each of the feature (e.g. Adverbs or Fragments) in *CorpusA* and *CorpusU* separately.

   Counting the frequency of most of the features is straight forward, as the *Stanford Parser* provides the POS categories of words and verb forms ("active" or

"passive") with almost perfect accuracy (86.36% accuracy, as documented by the study, [Klein & Manning, 2003]). It works as a bottom-up parser and chunks a complete sentence as "S" at the root of its parse tree, but whenever it encounters a fragment, it tags the root as "FRAG", "NP" or "SBAR" etc., that is anything other than "S". Thus, our feature extractor tool counts the number of times it finds the root of the generated parse tree different than "S", for example, as the total number of Fragments.

2. Dynamic Generation of Ambiguous Keywords:

We also made the feature extractor tool to dynamically generate a dictionary of ambiguous keywords from the training set. The words were all morphologically stemmed[1] by the stemmer that comes with Stanford Parser. Instead of counting only word-tokens, the keyword generator component counts (word-token, POS tag) pair, since the same word can be used in an unambiguous manner, attaining different POS tag [Wiebe et al., 2004]. Then the frequency of (word-token, POS tag) pairs are counted, and the ratio of their frequencies in *CorpusA* and in both *CorpusA* and *CorpusU* together are measured in a sorted hash-table, deriving their ranking of ambiguity automatically. For each keyword, we called this ratio its "Likelihood Ratio" (LR), denoting its strength of discrimination.

Before implementing the keywords generator component with the feature extractor program, We observed its outputs, and found that the there were many words (especially nouns) that simply by chance appeared more times in the CorpusA than in CorpusU. Thus, we also found ambiguous words in different POS categories had different weights in terms of LR. For example, ambiguous Adverbs tend have higher LR than ambiguous adjectives. Thus, it was clear that we couldn't rank all the keywords together based on their Likelihood Ratios.

We, therefore, coded the keywords generator component to consider only the words in the POS categories that generate ambiguities. These categories are: (1) JJ (Adjectives: words in this category include *high*, *low*, *small*, *good*,... etc.), (2) RB

[1] The morphological stemmer comes built-in with the Stanford parser [4]. It stems with the prior knowledge of the morphology of a word, and stems only to the point at which it retains its original POS class.

(Adverbs: words in this category include *highly*, *well*, *enough*, *strongly*,.... etc.), (3) MD (Modals: words in this category include *can*, *may*, *might*,.... etc. revealing optionality as ambiguity), and (4) DT (Determiners: words in this category include *all*, *some*, *any*.... etc. revealing ambiguous quantification). Then the keyword generator ranks the LR of the keywords for each of these POS categories separately. The user is then able to set a cut-off threshold[1] for each of the POS categories. The program then generates the list of those ambiguous keywords that have higher LR than the threshold. Consider Table 8 for example.

Keyword	POS	LR
All	DT	0.718
Any	DT	0.685
These	DT	0.658
Some	DT	0.575
Cut-off threshold = 0.5		
An	DT	0.462
Another	DT	0.386
No	DT	0.385
This	DT	0.362
A	DT	0.356

Table 8: Likelihood Ratio of the keywords of POS Category DT

In Table 8, only the first four keywords are saved in the list called **bad_DT**, since their LR is higher than the cut-off threshold.

Thus, the keyword generator adds four additional features to the feature list: *bad_JJ*, *bad_RB*, *bad_MD* and *bad_DT*, counting the number of times the enlisted ambiguous adjectives, adverbs, modals and determinants appear respectively.

3. Ranking all the features based on LR:

The feature extractor program not only extracts features, but also measures their ranking based on their Likelihood Ratios (LR), derived by the following formula:

[1] The threshold set for each of the four POS categories by the user is a real-valued number ranging from 0 to 1, where 0 accepts all words in a POS category, and 1 accept only those words that appeared in *CorpusA* but never in *CorpusU*.

$$\text{LR of Feature X} = \frac{\text{Frequency of X in CorpusA}}{(\text{Frequency of X in CorpusA} + \text{Frequency of X in CorpusU})}$$

The baseline LR is counted simply by the following formula:

$$\text{Baseline LR} = \frac{\text{Number of Words in CorpusA}}{(\text{Number of Words in CorpusA} + \text{Number of Words in CorpusU})}$$

Thus, a feature is kept as valid, iff its LR, derived by the above formula, is higher than the baseline LR, indicating the feature as a discriminating one. This idea was borrowed from the study of Wiebe *et al* [2004]. The LR ranking, denoting the strength of discrimination, of the features is shown in Table 9.

Feature	Description	Type	LR
bad_RB	Ambiguous Adverbs	Ambiguous Keywords	0.66316
bad_MD	Ambiguous Modals		0.63636
bad_JJ	Ambiguous Adjectives		0.51714
bad_DT	Ambiguous Determinants		0.47619
vbCount_in_p	Verbs in Parentheses	Syntactic Features	0.38709
tokenCount_in_p	Tokens in Parentheses		0.36958
paranthCount	Number of Parentheses		0.32174
fragment	Number of Fragments		0.27869
advCount	Number of Adverbs		0.25373
passCount	Number of Passives		0.22026
adjCount	Number of Adjectives		0.21914
Baseline	Baseline LR		0.21788

Table 9: Ranking of Features at the Sentence level

4. Extraction of Values of the Valid Features Only:

The *Feature Extractor* tool, thus, extracts the values of the valid features only, for each of the sentences in *CorpusA* and *CorpusU*, and saves their values in an specially formatted file, called the *training data file*. Since we used the open source machine learning workbench called *Weka* [Witten & Frank, 2005] to build our sentence classifier, we needed the training file to be formatted according to *Weka*'s requirements, in *Attribute-Relation File Format* (ARFF).

Segments from the ARFF training data file is shown in Figure 14. The header section of the file contains the @ATTRIBUTE declarations for each of the valid features (listed in Table 9) along with the data types of their values. There is also an additional @ATTRIBUTE declaration called "class", which can take any of the two nominal values: "Ambiguous" or "Unambiguous", denoting Annotator3's annotation of a particular sentence in our corpus. Each line under the @DATA declaration holds the features-values for a particular sentence in order of attribute declarations and ends with the corresponding annotation of the sentence, i.e. "Ambiguous" or "Unambiguous".

```
@RELATION sentence_corpus

@ATTRIBUTE  bad_DT  REAL
@ATTRIBUTE  bad_RB  REAL
@ATTRIBUTE  bad_MD  REAL
@ATTRIBUTE  bad_JJ  REAL
@ATTRIBUTE  vb_in_p  REAL
@ATTRIBUTE  tokn_in_p  REAL
@ATTRIBUTE  parentheses  REAL
@ATTRIBUTE  fragment  {TRUE,FALSE}
@ATTRIBUTE  adverbs  REAL
@ATTRIBUTE  passives  REAL
@ATTRIBUTE  adjectives  REAL
@ATTRIBUTE  class  {Ambiguous,Unambiguous}

@DATA
% ---from CorpusA
1,1,0,0,0,0,0,FALSE,2,0,1,Ambiguous
0,1,0,2,0,0,0,FALSE,1,0,2,Ambiguous
0,0,0,1,0,0,0,TRUE,0,0,2,Ambiguous
...
...

% ---from CorpusU
0,0,0,0,0,0,0,FALSE,0,0,1,Unambiguous
0,0,0,0,0,0,0,FALSE,0,0,0,Unambiguous
0,0,0,1,0,0,0,FALSE,0,0,1,Unambiguous
...
...
```

Figure 14: Training data file (in ARFF format) saved by the sentence-level *Feature Extractor*

All the sentences from *CorpusA* were processed first by our *Feature Extractor* tool. So, the first 236 lines under the @DATA declaration represented the feature-values

of all the 236 sentences of *CorpusA*, and therefore, had the "class" value "Ambiguous". Then, our *Feature Extractor* tool processed only the first 236 sentences of *CorpusU*. Thus, the following 236 lines of the *training data file* represented the only first 236 sentences of Corpus, and therefore, had the class value "Unambiguous". In this way, the *training data file* ensures equal distribution of data for "Ambiguous" and "Unambiguous" sentences, which is ideal for training.

5.5 Choice of Machine Learning Algorithm

We chose C4.5 decision tree learning algorithm [Quinlan, 1993] for the classification task. The two main reasons for this choice were: (1) decision trees could allow backtracking from a leaf to derive the cause of a particular classification, and C4.5 (revision 8), with its post-pruning feature, was the best open source decision tree learning algorithm we could avail. (2) The size of the corpus was not large enough for neural network algorithms to be trained on and attain better results.

For our next experiment, we used the machine learning workbench, called *Weka* [Witten & Frank, 2005] that provided a Java-based implementation of C4.5 (revision 8) algorithm, along with the necessary framework for training and evaluating our classifier.

5.6 Experiment and Results

In the experiment, our objective was to test the effect of using the valid features listed in Table 9 on the accuracy of a decision tree-based sentence classifier. For the experiment, we used the concatenation of *CorpusA* (reduced in size, as mentioned in section 5.2) and *CorpusU*, and considered all the 11 features listed in Table 9.

A batch program, called "*ishrarExperimentB.bat*", supplied with the distribution package, is used to run the experiment and display the result on-screen. The program uses the previously saved feature values of the 11 features, and trains the java implementation of C4.5 (revision 8) algorithm. The decision tree generated by the algorithm is shown in Figure 15.

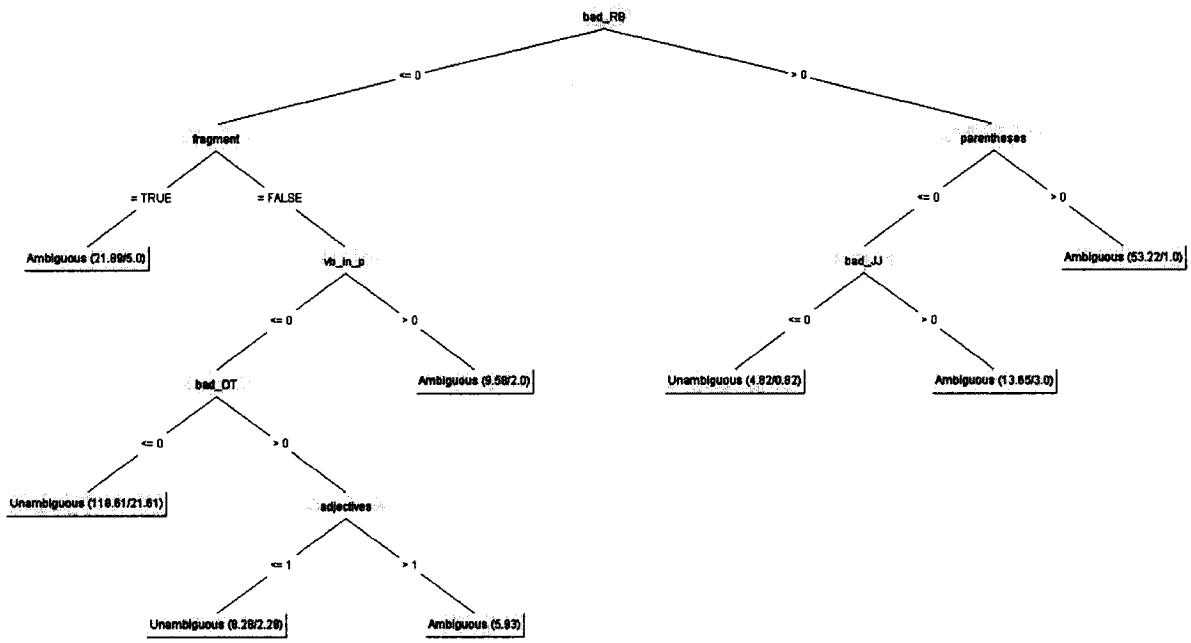

Figure 15: Decision Tree generated by C4.5 algorithm for Sentence Classification

The results of using C4.5 Algorithm to classify the sentences, using the values of 11 features are indicated in Table 10.

	Scheme	Correctly Classified Sentences	Incorrectly Classified Sentences	Kappa	Comment
Concatenation of CorpusA & CorpusU (Size = 236 + 236 = 472)	Training + Testing on same set	436 (92.37%)	18 (7.63%)	0.8475	Tree is of desirable characteristics, not sparse, and also not flat. None of the branches are wrongly directed. (see Figure 15)
	Cross-validation[1] (10 Folds)	418 (88.56%)	54 (11.44%)	0.7712	

Table 10: Results of using C4.5 algorithm to classify sentences

[1] *N* folds cross-validation technique first divides the corpus into *N* parts, and then uses one part for testing and the rest of the (*N*-1) parts for training the classifier. The process loops for *N* times, testing the classifier with each of the *N* parts of the corpus, and using the remaining (*N*-1) parts each time for training. [Witten & Frank, 2005]

Table 10 shows optimum result for our sentence classifier, where the accuracy was 92.37%, when both training and testing was done on the same corpus. We consider the results of 10-fold-cross-validation as the lower bound, as suggested by [Witten & Frank, 2005]. We found that the lower bound accuracy of the classifier was also very high (88.56%). The careful annotation work at the sentence-level and usage of precise features have also achieved a very high degree of agreement with the human annotations (Kappa index 0.8475, when both training and testing were done on the same corpus, and 0.7712, when 10-fold-cross-validation was performed).

We also found that the classification tree generated by the C4.5 algorithm was of exact desirable characteristics, where not all of the 11 features were selected by the algorithm. Rather, only the ones that are enough to discriminate the sentences were chosen. The resultant tree after training is shown in Figure 15. It should be noted that this tree was dynamically generated. Therefore, with the introduction of new training data, the classifier is able to generate new decision trees.

5.7 Conclusion

Our results with this initial experiment affirm that the task of detecting ambiguity in terms of Surface Understanding is indeed doable by means of currently available NLP tools and text classification techniques. The accuracy of our sentence classifier establishes its applicability in practical fields, where ambiguity is detected at the sentence level. But, since our primary concern was to detect ambiguity at the discourse level, we continued with our work of building a more powerful classifier that could best emulate the decisions of our human annotators by classifying a discourse based on its ambiguity at the level of surface understanding.

Chapter 6

Discourse-level Classification

6.1 Methodology

We again employed a text classification technique for detection of ambiguity in surface understanding, but this time, at the discourse-level. We used our discourse-level annotated corpus for this purpose. Each instance of this corpus is a *passage* of a problem description (see section 4.3.1, for our definition of *passage* in this thesis). Thus, the scope of discourse that we took into consideration for ambiguity detection was limited to one **passage** only.

Discourse-level classification was performed in the following two distinct ways, so that they can be compared afterwards in terms of efficiency:

(a) Using Discourse Features
(b) Using The Sentence Classifier

To perform discourse-level classification using discourse features, we first tried to identify a list of *extractable* lexical and syntactic features in a discourse that can make it ambiguous in terms of surface understanding. Our *Feature Extractor* tool, using the *Stanford Parser* [Klein & Manning, 2003], then extracted the values of all those features in each of the instances (passages) in our corpora, and trained our classifier with the feature values and the corresponding annotation of the instances. Figure 16 illustrates the steps of the classification.

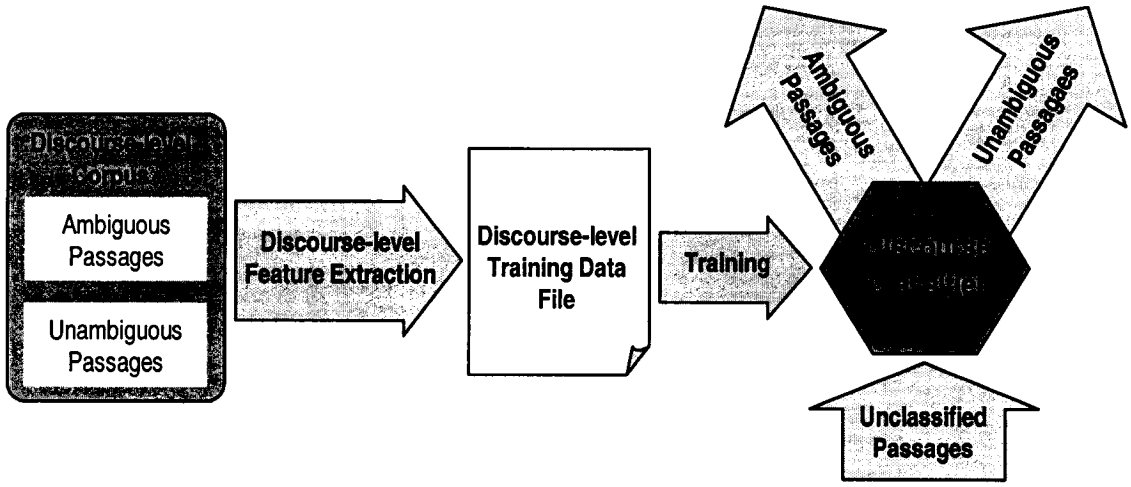

Figure 16: Discourse-level Classification using Discourse Features

We also intended to build another discourse-level classifier using our sentence classifier. Figure 17 illustrates the process. Here, the sentence classifier is used to count the number of ambiguous sentences of in each of the instances (passages) in our corpora, and classify an instance based on the density of ambiguous sentences (along with two other discourse features, which were, although, ignored later by our classification algorithm on the basis of our training data file) and the corresponding annotation of the instance.

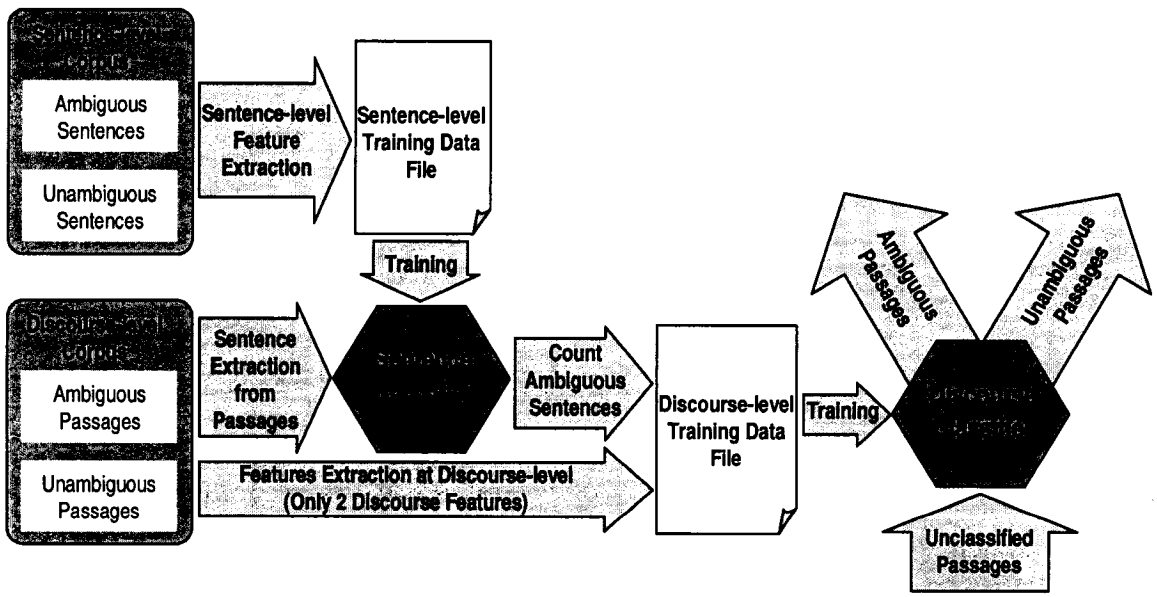

Figure 17: Discourse-level Classification using the Sentence Classifier

The following sections explain our two ways of performing discourse-level classification in details, and discuss the experiments and results with comparisons of efficiency.

6.2 Using Discourse Features

6.2.1 Pre-processing

As discussed in chapter 4, human annotators previously annotated 165 instances of passages from 25 problem descriptions of our discourse-level corpus by giving scores in the range of 0 to 10. But two issues persisted with the annotations: 1) the gold-standards of some instances, although measured by the decision of the majority of the annotators, had a considerable number of annotators disagreeing with it; 2) If the decision threshold on the score is set to 5 (as stated in the standard interpretation of the scores given to the annotators — see section 4.4.1.3 for details), the size of the corpus becomes unequally divided into two sets: Ambiguous (6%) and Unambiguous (94%), which is *unfavorable* for training any machine learning algorithm. To deal with these issues, the following two preprocessing tasks are undertaken:

1. **Reduction of the size of the corpus:**

   If the gold-standard of an instance agreed with the annotations of at least two annotators more than the number of annotators disagreeing to it, only then the sample was included in the new corpus for detecting ambiguity. Table 11 presents this idea with a hypothetical example, where instance 1 would be kept, but both instance 2 and 3 would be excluded from the corpus following the aforesaid rule.

	Annotator 1		Annotator 2		Annotator 3		Annotator 4		Gold Standard (median)		Keep / Exclude
	Score	Anno-tation	Score	Anno-tation	Score	Anno-tation	Score	Anno-tation	Score	Anno-tation	
Instance 1	8	U	8.5	U	10	U	4	A	8.25	U	Kept
Instance 2	4	A	9	U	8	U	3	A	6	U	Excluded
Instance 3	7.6	U	6	U	4	A	5		5.5	U	Excluded

Table 11: A hypothetical example of the preprocessing task

Here, instance 2 can easily be spotted having a weak gold-standard decision with equal number of annotators annotating the instance as "Ambiguous" and "Unambiguous", and the rule rightly excludes the instance. According the rule, instance 3 also gets excluded, because two annotators annotated the instance as "Unambiguous", which is only one more than the number of annotator (only one) annotated as "Ambiguous", thus, making the gold-standard decision comparatively weaker than the decision of instance 1. Thus, we excluded 17 such weak instances from our corpus using the rule, reducing its size to 165. Table 12 shows the improvements in the kappa index resulting from the reduction of the size of the corpus.

	Sample Size	Inter-Annotator Agreement	
		Average Pair-wise Agreement with Gold-standard (in Kappa)	Multiple Annotators' Combined Agreement Measure (in Kappa)
Old Corpus	165	0.6661	0.4492
New Corpus	148	0.737	0.5441

Table 12: Improvements in Kappa after reduction in corpus size

2. Raising the decision threshold of the Gold-Standard:

Instead of using a decision threshold of 5, we used 8.5, meaning that if an instance has the median of the annotators' scores greater than 8.5, the instance is labeled as "Unambiguous", and otherwise, it is labeled as "Ambiguous". This divided the corpus in two sets which are nearly equal in size. The resultant new distribution of the corpus is shown in Figure 18.

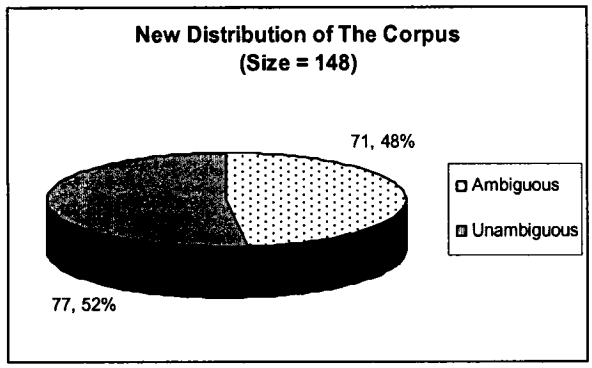

Figure 18: Distribution of the Corpus after preprocessing tasks mentioned in section 6.2.1

6.2.2 Our Initial List of Features

We have discussed some of the characteristics of natural language text in section 3.4 that are presented in different studies as potential *clues* revealing ambiguity, and are extractable by a syntactic parser (equipped with a POS-tagger). The annotation guideline that we prepared in consultation with the experts (see section 4.4.1.2) also points out some characteristics of texts that reveal ambiguity and are also extractable by the aforesaid parser (e.g. frequency of *Passive Verbs*).

With this end in view, we first tried to identify a list of *extractable* lexical and syntactic features in a passage that can make it ambiguous. Higher frequency of each of these features in an instance of a passage in our corpora apparently has the potential to induce ambiguity in that passage in terms of surface understanding. These features are listed below:

1. *Number of Words per sentences in a passage*
2. *Number of Adjective* in a passage*
3. *Number of Adverbs* in a passage*
4. *Number of Passive Verbs* in a passage*
5. *Number of Parentheses* in a passage*
6. *Total number of Tokens* inside (or, between) all pairs of parentheses in a passage*

7. *Total number of all forms of Verbs* inside (or, between) all pairs of parentheses in a passage*
8. *Number of Punctuations* in a passage*
9. *Number of Conjunctions* in a passage*
10. *Number of Adverbs modifying Verbs* in a passage*
11. *Number of Adverbs modifying Adjectives* in a passage*
12. *Number of Cardinalities (quantifying numbers)* in a passage*
13. *Number of Degrees* in a passage*
14. *Number of Tables* in a passage*
15. *Number of Images* in a passage*

The frequency of features above, marked with asterisk (*), are *normalized* (i.e. arithmetically divided) by the total number of words in a passage.

From this list, features 1, 2, 3 and 4 are also used by [Fabbrini et al., 2001; Gnesi et al., 2005; Wilson, 1997]; and, the ambiguity characteristics of features 1, 2, 3, 4, 6, 7, 14, 15 are also mentioned in our annotation guideline, described in section 4.4.1.2.

6.2.3 Sentence Parsing and Feature Extraction

We developed a *Feature Extractor* program to extract the values of all the 15 features previously mentioned. The program first detects sentence boundaries to extract one sentence at a time, so that it can feed the sentences one-by-one to *Stanford Parser* [Klein & Manning, 2003] for POS tagging and syntax parsing. The values of the features are then counted for each instance of passages from our discourse-level corpus.

Table 13 shows the 15 features ranked according to the absolute values of correlation between their individual values that we extracted and the medians of the annotators' surface understanding scores for all instances:

Feature	Correlation with S.U. Score
words /sentence **[W]**	0.2814
adverbs (modifying adjectives) /word **[AA]**	0.2715
adverbs /word **[AD]**	0.2141
adjectives/word **[AJ]**	0.1497
tokens (in parentheses) /word **[TP]**	0.1372
passives/word **[PS]**	0.1372
adverbs (modifying verbs) /word **[AV]**	0.1328
punctuations/word **[P]**	0.1205
degrees /word **[D]**	0.1165
verbs (in parentheses) /word **[VP]**	0.112
images /word **[I]**	0.0737
parentheses /word **[Pr]**	0.0463
cardinals /word	0.0406
tables /word **[T]**	0.0257
conjunctions /word **[C]**	0.0235

Table 13: Absolute values of correlation between "each of the values of the initial feature list and the median of all the annotators' Surface Understanding scores" pairs for all instances

Thus, Table 13 shows that the number of words/sentences is a very important feature, but surprisingly, the number of conjunctions is not.

6.2.4 Training data file

The *Feature Extractor* program stores the values of all the features for all instances in an specially formatted file, called the *training data* file. Here, again, since we used the open source machine learning workbench called, *Weka* [Witten & Frank, 2005] to build the classifier, we needed the *training file* to be formatted according to *Weka*'s requirements, in

Attribute-Relation File Format (ARFF). A segment of the ARFF training data file is shown in the following figure:

```
@RELATION discourse_corpus_1

@ATTRIBUTE  words_per_sentences  REAL
@ATTRIBUTE  adverbs  REAL
@ATTRIBUTE  adv_mod_adj  REAL
@ATTRIBUTE  adv_mod_vb  REAL
@ATTRIBUTE  adjectives  REAL
@ATTRIBUTE  conjunctions  REAL
@ATTRIBUTE  degrees  REAL
@ATTRIBUTE  cardinalities  REAL
@ATTRIBUTE  passives  REAL
@ATTRIBUTE  punctuations  REAL
@ATTRIBUTE  tokens_in_paranth  REAL
@ATTRIBUTE  verbs_in_paranth  REAL
@ATTRIBUTE  parantheses  REAL
@ATTRIBUTE  tables  REAL
@ATTRIBUTE  images  REAL
@ATTRIBUTE  class  {Ambiguous,Unambiguous}

@DATA
16,0.0625,0,0.0625,0.09375,0.0625,0,0,0,0.0625,0,0,0,↵
0,0,Unambiguous
16.666666,0.02,0,0.02,0.1,0.04,0,0.04,0.02,0.06,0.02,↵
0,0.02,0,0,Unambiguous
8.75,0.057142857,0,0.057142857,0.14285715,0.08571429,↵
0,0,0,0.028571429,0,0,0,0,0,Ambiguous
9.571428,0,0,0,0.05970149,0,0,0,0,0.014925373,0,0,0,0↵
,0,Unambiguous
32,0.015625,0,0,0.0625,0.015625,0,0,0.015625,0.03125,↵
0.046875,0,0.015625,0,0,Ambiguous
...
...
```

Figure 19: Training data file (in ARFF format) saved by the discourse-level *Feature Extractor* mentioned in section 6.2.3

Thus, the output of the *Feature Extractor* program is an ARFF training file, which is an ASCII text file describing the instances of our corpus with corresponding feature values and their annotations. In Figure 19, we see that the header section of the training data file contains the metadata of our corpus, holding names of all the 15 features as `@ATTRIBUTE` declarations of real-valued features, and also the classification outputs as an additional `@ATTRIBUTE` declaration, called "class", with the nominal values "Ambiguous" and "Unambiguous". The `@DATA` section holds in each line the *comma-separated-values* (CSV)

of all the 15 features in order of declaration, where each line corresponds to one instance in our corpus. And, at the end of each line, the nominal value of "ambiguous" or Unambiguous" denotes the corresponding annotation of the instance.

6.2.5 Choice of Classification Algorithm

We again chose C4.5 decision tree learning algorithm [Quinlan, 1993] for the classification task. The reasons for this choice were same as mentioned in section 5.5 from the previous chapter.

We used the Java-based implementation of C4.5 (revision 8) algorithm available with *Weka* [Witten & Frank, 2005] for the experiments described next.

6.2.6 Experiments and Results

We carried out the following experiments using the feature values for each instances extracted by the feature extractor, and both the old and newly preprocessed discourse-level corpora mentioned above.

6.2.6.1 Experiment A1

In this initial experiment, we intended to test the effect of reducing the corpus size on the accuracy of a decision tree-based classifier. Both the old discourse-level corpus (of size 165) and the newly reduced discourse-level corpus (of size 148) after preprocessing were used in this experiment. The experiment considered all the 15 features that were mentioned in Table 13. The batch program, "*ishrarExperimentA1.bat*", was used to run the experiment and display the result on-screen. The program uses the previously saved feature values and trains the java implementation of C4.5 (revision 8) algorithm. It then tests the classifier once using the same corpus, which it is trained on, and the other time, using 10-fold-cross-validation technique. Their results, before and after the reduction of corpus set **(a total of 15 initial features were used)** are shown in Table 14.

	Scheme	Correctly Classified Instances	Incorrectly Classified Instances	Kappa	Remarks on the decision tree
Old Corpus Set (Size = 165)	Training + Testing on same corpus set	132 (80%)	33 (20%)	0.5971	Tree is very sparse, same features considered on different nodes
	Cross-validation (10 Folds)	94 (56.97%)	71 (43.03%)	0.1416	
New Corpus Set (Size = 148)	Training + Testing on same corpus set	138 (93.24%)	10 (6.76%)	0.8648	Tree is even more sparse, same features considered on many nodes
	Cross-validation (10 Folds)	78 (52.7%)	70 (47.3%)	0.0566	

Table 14: Results of using C4.5 algorithm on initial 15 features

The results in Table 14 show that reducing the size of the corpus significantly improves the accuracy (percentage of correctly classified instances increased from 80% to 93.24%, when training and testing is performed on the same corpus set). The Kappa index, measuring the degree of agreements between the human annotations and automatic classification also increased, when the same corpus set was used on training and testing.

But, having too many fine-grained features (15 features) supplied to the classification algorithm over-fits the data by generating a very sparse tree, and thus, attains a poor result with a 10-fold-cross-validation scheme. This issue led to the refinement of the list of features in the following experiments. We can assume that the results, when using the same training set as the testing set, represents the upper-bound of the classifier's accuracy, whereas, results of 10-fold-cross-validation can serve as the lower-bound.

Therefore, we continued using only the new preprocessed discourse-level annotations for the rest of the experiments.

6.2.6.2 Experiment A2

In this experiment, we intended to test the effect on the accuracy of the same classifier of introducing a new feature, along with the 15 initial features used in Experiment A1. For this, our feature extractor program is updated to calculate the frequency of *Uniques*, which are the number of words that appear once in a discourse, for each passage. The frequency is then normalized by the number of total words in the passage. We used this normalized frequency as our new feature, called ***Uniques/ word***. This feature alone attained a correlation of **0.3088** with the surface understanding scores of human annotators, which is the highest among compared to that of all other features. This feature was also used to detect ambiguity in requirements by the study of Wasson *et al* [2005].

For the experiment, we used the pre-processed discourse-level corpus (of size 148), and considered 16 features now, i.e. all the 15 features that were mentioned in Table 13, plus the new feature ***Uniques/ word***. The batch program, *"ishrarExperimentA2.bat"*, supplied with the distribution package, was used again to run the experiment and display the result on-screen. Like before, the program uses the previously saved feature values and trains the java implementation of C4.5 (revision 8) algorithm. It then tests the classifier once using the same corpus, which it is trained on, and the other time, using 10-fold-cross-validation technique. The results **(a total of 16 features were used)** are shown in Table 15.

	Scheme	Correctly Classified Instances	Incorrectly Classified Instances	Kappa	Remarks on the decision tree
New Corpora Set (148)	Training + Testing on same set	142 (95.95%)	6 (4.05%)	0.9188	Tree remains very sparse with same features considered on different nodes
	Cross-validation (10 Folds)	88 (59.46%)	60 (40.54%)	0.1914	

Table 15: Results of using C4.5 algorithm on initial 15 features + (Uniques /word) as a feature

The results in Table 15 shows that introducing (*Uniques* /word) as a feature significantly improves the accuracy (percentage of correctly classified instances increased from 93.24% to 95.95%), when training and testing is performed on the same corpus set). The Kappa index,

measuring the degree of agreements between the human annotations and automatic classification also increased, when the same corpus set was used for both training and testing. In this experiment, we find that the statistics for cross-validation also improved as Kappa value increased from 0.0566 to 0.1914.

But again, having too many fine grained features (16 features this time) over-fitted the data by generating a very sparse tree, and thus, attains a poor result with a 10-fold-cross-validation scheme. This issue led to further refinement in the list of features in the following experiments.

Therefore, we decided to include **Uniques /word** as a valid feature in the previous list of features.

6.2.6.3 Experiment A3

Here, we wanted to test the effect on the classifier's accuracy of reducing its number of features by first manually combining them into four major feature-groups, and then using them for training the classifier. For this, different combinations (functions) of the initial features were generated by adding them with manually set weights to group them up into different major features. These features were then ranked according to their individual correlation with the median of the annotators' scores. The first four major features that achieved the highest correlation with the median of the annotators' scores, were selected. They are shown in Table 16.

Raw Feature	Correlation with S.U. Score (absolute value)	Combined As	Function (manually defined)	Correlation with S.U. Score (absolute value)
uniques /word **[U]**	0.3088	**Uniques**	U	0.3088
words /sentence **[W]**	0.2814	**Length**	(W/40) + TP + C + (Px2)	0.328
tokens (in parentheses) /word **[TP]**	0.1372			
conjunctions /word **[C]**	0.0235			
punctuations/word **[P]**	0.1205			

adverbs /word **[AD]**	0.2141	**Modifiers**	AD + (AAx10) + (AVx3) + AJ + (Dx5)	0.321
adverbs (modifying adjectives) /word **[AA]**	0.2715			
adverbs (modifying verbs) /word **[AV]**	0.1328			
adjectives/word **[AJ]**	0.1497			
degrees /word **[D]**	0.1165			
passives/word **[PS]**	0.1372	**Ambiguous Verbs**	PS + (VPx2)	0.1766
verbs (in parentheses) /word **[VP]**	0.112			

Table 16: 12 of the initial features manually combined into four major features

For the experiment, we used the same pre-processed discourse-level corpus (of size 148), and only 4 major manually combined features. The batch program, *"ishrarExperimentA3.bat"*, was used to run the experiment and display the result on-screen. Like before, the program uses the previously saved feature values of the four features, and trains the java implementation of C4.5 (revision 8) algorithm. It then tests the classifier once using the same corpus, which it is trained on, and the other time, using 10-fold-cross-validation technique. Their results are shown in Table 17.

	Scheme	Correctly Classified Instances	Incorrectly Classified Instances	Kappa	Comment
New Corpora Set (148)	Training + Testing on same set	106 (71.62%)	42 (28.38%)	0.4284	Tree is very flat and simple with only 4 nodes, but few wrong directions show noise in the training data (see Figure 20)
	Cross-validation (10 Folds)	85 (57.43%)	63 (42.57%)	0.143	

Table 17: Results of using C4.5 algorithm trained with 4 major features

The results in Table 17 shows that the classifier loses accuracy when the number of features is reduced to four, by combining them with weights set manually. But we also find that the accuracy of the classifier in the 10-fold-cross-validation results, although was reduced, but not significantly. This points out the fact that the tree was not sparse and was not over-fitted to the data.

But a new issue arose at this point. If we visualize the resulting tree after training with the preprocessed corpus set, we find two branches were wrongly directed. For example, if the value of the combined feature, called "modifier" increases, the classification outcome should tend to be "Ambiguous" (as it is inversely proportional to annotators' scores by definition). But C4.5 algorithm generates a decision-tree (as shown in Figure 20) that assumes the "modifier" feature to be directly proportional to the annotators' scores, because of the existence of many modifiers in the training data that never contribute to ambiguity at the level of surface understanding. Thus, it generates a wrong directional branch, setting the rule "*if the value of the* ***modifier*** *feature is higher than 0.169231 than an instance is to be classified as* ***Unambiguous***". The same problem was found for the feature called, "Ambiguous Verbs (a_verbs)".

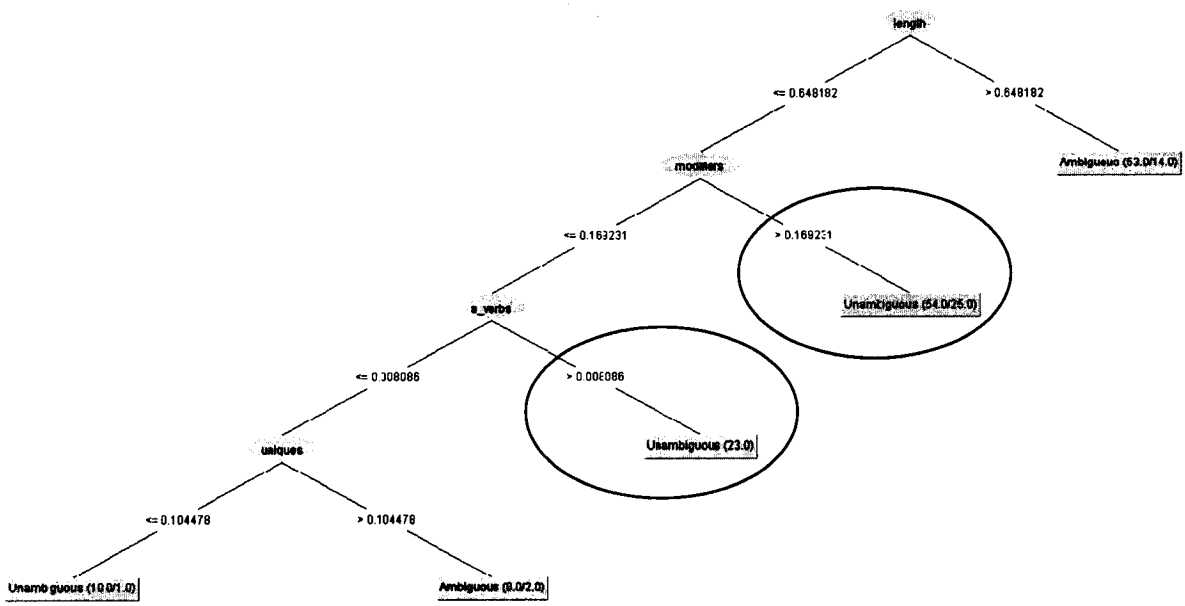

Figure 20: Decision Tree of C4.5 using 4 major manually combined features (wrong directional branches resulted from noise are shown in circle)

This issue points to the fact that discourse-level annotation data, along with the values of the features themselves represented "noisy" data.

6.2.6.4 Experiment A4

Here, we intended to test the effect on the classifier's accuracy of first reducing the number of features by combining them into four major feature-groups automatically with linear regression, and then using them for training the classifier. For this, the fine-grained features that were related by definition were first manually grouped into four categories. Then the least median squared linear regression method [Rousseeuw & Leroy, 1987] was used to find the combinations (functions) of the fine-grained features, where the weights were set automatically, so that they attain a high correlation with annotators' scores in surface understanding. The resulting four features are shown in Table 18.

Raw Feature	Correlation with S.U. Score (absolute value)	Combined As	Function (defined by linear regression)	Correlation with S.U. Score (absolute value)
uniques /word **[U]**	0.3088	**Uniques**	U	0.3088
words /sentence **[W]**	0.2814	**Length**	(0.0311xW) + (1.7807xC) + (0.1619xP) + (3.0231xTP) - (10.7588xPR)	0.3286
tokens (in parentheses) /word **[TP]**	0.1372			
parentheses /word **[PR]**	0.0463			
conjunctions /word **[C]**	0.0235			
punctuations/word **[P]**	0.1205			
adverbs /word **[AD]**	0.2141	**Modifiers**	(-3.0476xAD) + (42.1012xAA) + (6.31xAV) + (1.3115xAJ) + (8.9913xD)	0.3158
adverbs (modifying adjectives) /word **[AA]**	0.2715			
adverbs (modifying verbs) /word **[AV]**	0.1328			
adjectives/word **[AJ]**	0.1497			
degrees /word **[D]**	0.1165			
passives/word **[PS]**	0.1372	**Passives**	PS	0.1372

Table 18: Twelve of the initial features combined by linear regression into four major features

For the experiment, we used the same pre-processed discourse-level corpus (of size 148), and 4 major features mentioned above. A batch program, called *"ishrarExperimentA4.bat"*, supplied with the distribution package, is used to run the experiment and display the result on-screen. Like before, the program uses the previously saved feature values of the four features, and trains the java implementation of C4.5 (revision 8) algorithm. It then tests the classifier once using the same corpus, which it is trained on, and the other time, using 10-fold-cross-validation technique. Their results are shown in Table 19.

	Scheme	Correctly Classified Instances	Incorrectly Classified Instances	Kappa	Comment
New Corpora Set (148)	Training + Testing on same set	106 (71.62%)	42 (28.38%)	0.4364	Tree is very flat and simple with only 4 nodes, but one wrongly directed branch show noise in the training data (see Figure 21)
	Cross-validation (10 Folds)	88 (59.46%)	60 (40.54%)	0.1807	

Table 19: Results of using C4.5 algorithm trained with 4 major features

The results in Table 19 show that the classifier's accuracy slightly improved in this case. But if we examine the tree generated by the C4.5 algorithm (as shown in Figure 21), we again find one wrongly directed branch representing noise in the data.

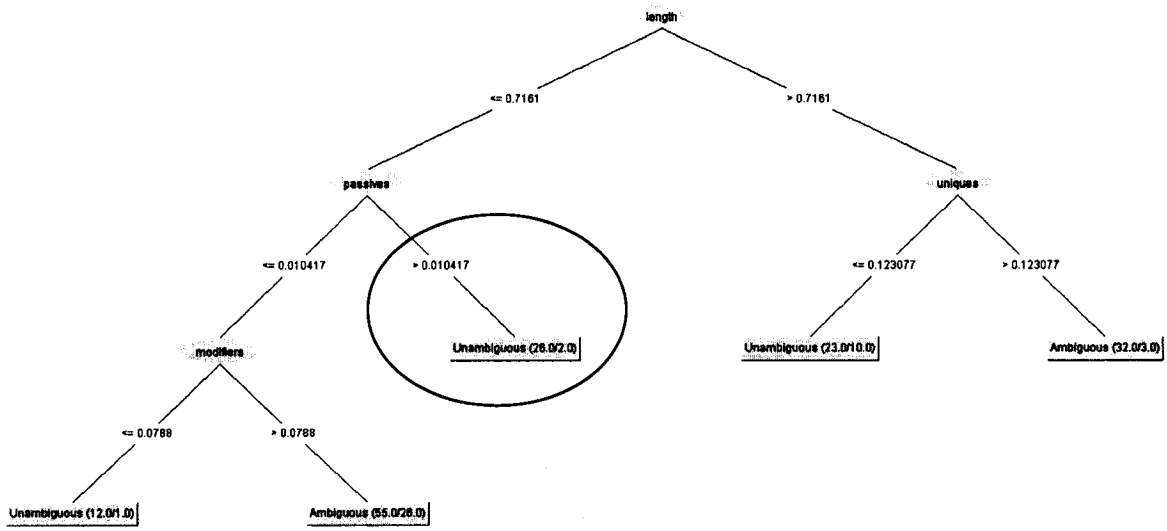

Figure 21: Decision Tree of C4.5 using 4 major linear regression features (one wrong directional branch resulted from noise is shown in circle)

6.2.7 Analysis

The previous experiments have shown that the accuracy of the discourse-level classifier using discourse features is not high enough for it to be realistically applied in practical fields to detect ambiguity.

To investigate the reasons of having noise in our data, we can first consider an example of passage from our corpus, given Figure 22.

```
A cell in a cellular telephone network handles all the calls of mobile
phones in the area it covers. To do this, it transmits identifying
information on a beacon radio frequency and handles the "traffic" on
other frequencies. When a network is initially built, the area that is
covered by each cell is very large. When the number of subscribers
increases, capacity is added by splitting cells. This results in more
and smaller cells, offering more total capacity. This technique makes
cellular networks highly scalable. In busy areas like cities, cell
diameters are measured in metres, in rural areas the diameters can be
up to 64 km. The only limit is the interference of cells that reuse the
same frequency. Networks that use digital signals instead of analog
between the cell and the mobile phone have an advantage because the
interference levels can be considerable higher at the same quality.

(IMAGE)

One of the most complicated aspects of a cellular network is the
planning of the frequencies. This is almost a black art and is usually
handled by special planning programs. These programs contain prediction
algorithms that take into account the terrain that the cell covers.
```

Figure 22: An example of a passage from our corpus

The median of our annotators' scores for this particular passage is 7. Thus, after the pre-processing mentioned in section 6.2.1, the annotation for this instance is selected as "Ambiguous" (since, the median score less than 8.5).

Now, our Feature Extractor program extracted the following feature values for this instance:

words /sentence	adverbs /word	adjectives /word	tokens-in-parentheses /word	passives /word	...
16.4167	0.0355	0.0964	0	0.0304	

Table 20: Feature Values of Example in Figure 22

Now, we have seen from Table 13 that "*words/sentence*" attains the highest rank of all features (which is, although, later on superseded by the feature: "*Uniqes /word*" — see experiment A.2 of section 6.2.6), indicating it to be the most discriminating feature of that list. We also find that 62 instances have the value of *words/sentence* less than 19.5 and they are all annotated as "Unambiguous", while 35 instances have the value of *words/sentence* higher than 19.5, and they are all annotated as "Ambiguous". But, there exist 51 other instances, like the one shown in Figure 22 earlier, which contradict this rule and are annotated as "Ambiguous" and "Unambiguous" considering other features.

For example, in Table 20, the instance has the value of words/sentence = 16.4167, which is less than 19.5 mark, yet, the annotation of the instance in our corpus is "Ambiguous". Although it has one sentence containing 35 words and another containing 30 words (see Figure 22), on average these chances of ambiguity get ignored.

This is the primary problem of using our list of generalized syntactic and lexical features at discourse-level, when many strong features lost their power of discrimination after they were normalized in the discourse. The reason is that irregular presence of Unambiguous characteristics in greater number in a text reduces the chances of an Ambiguous characteristic to be acknowledged. Such irregularities increased noise in our data that eventually led our C4.5 decision-tree-based classifier to perform poorly after training.

Hence, we looked forward to using the sentence classifier, discussed in chapter 5, to isolate the scope of the syntactic features and redo task of discourse-level classification to analyze its performance at discourse-level.

6.3 Using the Sentence Classifier

6.3.1 Preprocessing

As discussed in section 5.1, we have used *Annotator3*'s sentence-level annotation (reviewed and refined by both *Annotator1* and *Annotator3*) for the purpose of building the training corpus of sentence classifier. Our goal was to use this same classifier for the task of discourse classification. To achieve this goal, we undertook the following two preprocessing tasks:

a) Selecting the Scores of the Gold-Standard:

The Annotation Guideline, described in section 4.4.1.2, permitted all of our annotators to have freedom in scoring and use their own scoring technique in selecting a higher or lower score while annotating the discourse-level corpus. This allowed an annotator to have a different mental model than the other while scoring an instance in terms of Surface Understanding. And, we wanted to seek out the Annotator, whose scoring technique would conform the most to the detection of ambiguous sentences using our sentence classifier.

When we interviewed the annotators to know their scoring technique, we identified that *Annotator3*'s scores for discourse-level annotation precisely followed the sentence-level annotation that she had done beforehand — a technique that others never used. Our inspection on the corpus also supported this fact that she mostly followed a rule of reducing 1 point from her score for every ambiguous sentence that she could find in an instance. Thus, for using our sentence classifier, which had already been trained with *Annotator3*'s sentence-level annotations, our best option was to use *Annotator3*'s discourse-level scores as the decisive scores of the gold-standard of our corpus, and build a discourse-level classifier using the annotations. It should also be mentioned here that *Annotator3* (Dr. Ormandjieva) is the only expert in the field of Requirements Specification in our group of annotators.

b) Raising the Decision Threshold:

For our new gold standard, we raised the decision threshold to 9, so that, we could divide the discourse-level corpus in two nearly equal half based on *Annotator3*'s scores. Therefore, if the score was higher than 9, the instance was held to be annotated as "Unambiguous", and, otherwise, "Ambiguous".

The resultant distribution of the corpus after the preprocessing tasks is shown in Figure 23.

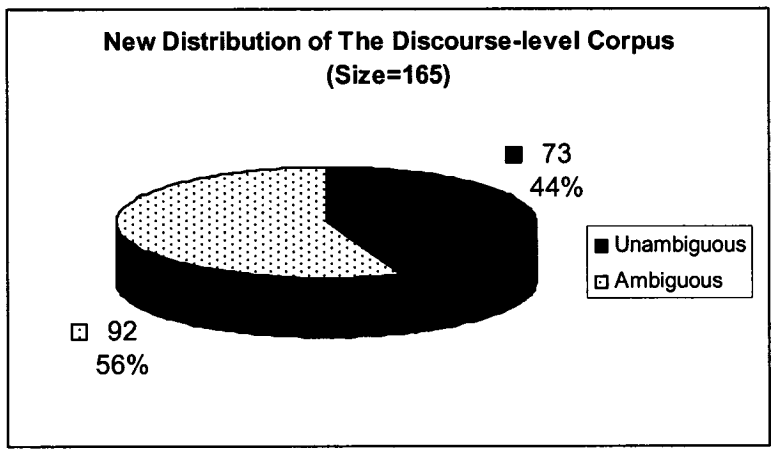

Figure 23: Distribution of the Corpus after preprocessing tasks mentioned in section 6.3.1

6.3.2 Ambiguous Sentences as a Discourse Feature

We developed a *Feature Extractor* program that first trains the sentence-level classifier with the *Sentence-level training data file*, which had been created earlier by the tool described in section 5.4. The program then reads each passage or instance of the discourse-level corpus and extracts sentences from them. The sentences are then syntactically parsed, one at a time, using the *Stanford Parser* [Klein & Manning, 2003], and the values of the required features (see Table 9) are counted and fed into the trained sentence classifier. If the sentence is classified as an ambiguous sentence by the sentence classifier, a counter increases counting the frequency of ambiguous sentences for a particular instance in our discourse-level corpus.

We found that the absolute value of the correlation between the number of ambiguous sentences in the passages of our corpus and the corresponding scores of the passages given *Annotator3* is **0.62506**, which is higher than any discourse features used in the experiments of section 6.2.6 (compare with the values mentioned in Table 13). Figure 24 shows the distribution of ambiguous sentences in our discourse-level corpus with 165 instances of passages:

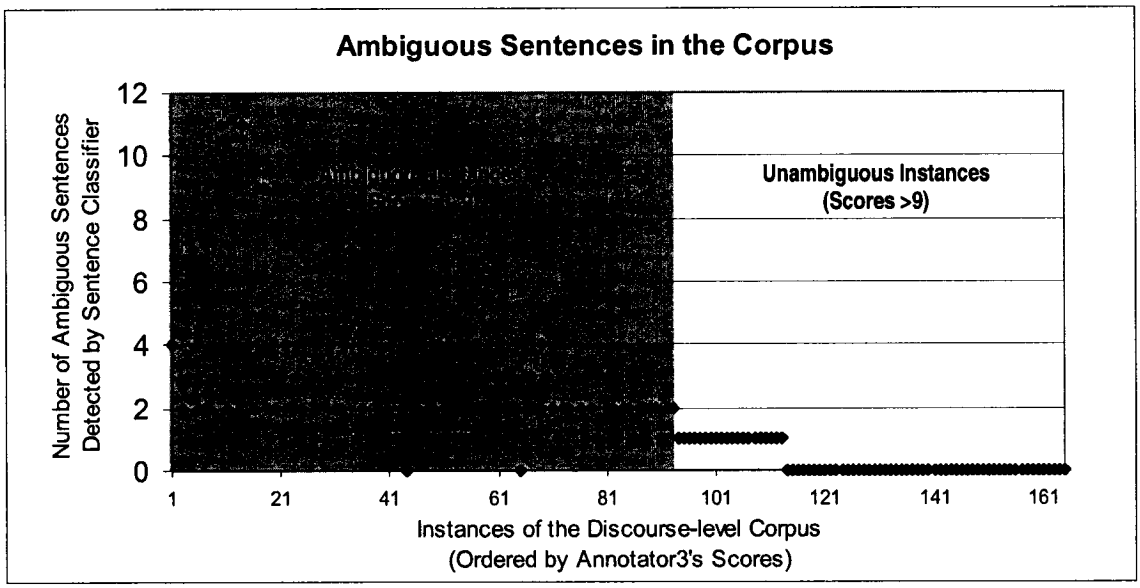

Figure 24: Number of Ambiguous Sentences detected by the Sentences Classifier in the instances of the Discourse-level Corpus (Instances are sorted by *Anootator3*'s scores during annotation)

Therefore, we selected our new discourse feature as the density of ambiguous sentences (number of ambiguous sentences divided by the total number of sentences) in an instance. For the next experiment we also supplied the feature values two of the strongest discourse features used in our previous experiments (discussed in section 6.2.6) — "***Uniques /word***" and "***words /sentences***", along with the new "***ambiguous-sentences /sentences***" features, to the C4.5 decision-tree learning algorithm [Quinlan, 1993] to check if the resultant decision tree includes the previous features as well.

Thus, all these feature values for each of the instances, with their corresponding annotations are saved in the *Discourse-level training data file* (which is in *ARFF* format, similar to the one mentioned in section 6.2.4) by the *Feature Extractor* program. The Discourse-level

classifier program reads this file to train the C4.5 algorithm using *Weka*'s framework [Witten & Frank, 2005]. The following figure shows a part of training file:

```
@RELATION discourse_corpus_2

@ATTRIBUTE  ambiguous_sent_per_sentence  REAL
@ATTRIBUTE  uniques_per_word  REAL
@ATTRIBUTE  words_per_sentence  REAL
@ATTRIBUTE  class  {Ambiguous,Unambiguous}

@DATA
0.2,0.603603604,22.2,Ambiguous
0.086956522,0.334916865,18.30434783,Ambiguous
0.2,0.242424242,19.8,Ambiguous
0.235294118,0.194581281,23.88235294,Ambiguous
0.166666667,0.154811715,19.91666667,Ambiguous
0,0.138888889,18,Unambiguous
0.090909091,0.237837838,16.81818182,Ambiguous
0,0.385964912,57,Unambiguous
0.25,0.171717172,24.75,Ambiguous
0.25,0.151394422,31.375,Ambiguous
0.263157895,0.146981627,20.05263158,Ambiguous
0,0.048780488,20.5,Unambiguous
...
...
```

Figure 25: Training data file (in ARFF format) saved by the discourse-level *Feature Extractor* mentioned in section 6.3.2

6.3.3 Building the Discourse-level Classifier

We again chose the C4.5 (revision 8) decision-tree learning algorithm [Quinlan, 1993] for the purpose of building the discourse-level classifier. The *Weka* machine learning workbench [Witten & Frank, 2005] provided the necessary framework for training, and evaluating our classifier.

We used the *Discourse-level training data file* saved by our *Feature Extractor* program, as mentioned in the previous section, and trained our classifier with the file. The classifier, therefore, requires three feature values to be extracted from a passage of unknown status, so that it can predict its classification as the nominal values: "Ambiguous" or "Unambiguous".

The decision-tree generated by the aforesaid C4.5 algorithm after training is shown in Figure 26.

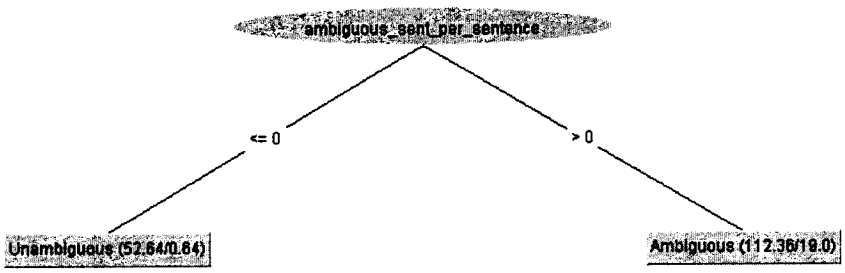

Figure 26: Decision tree generated by C4.5 learning algorithm [Quinlan, 1993] after training with the feature-values mentioned in section 6.3.2

Figure 26 shows the discrimination power the sentence classifier has in classifying discourse. The tree contains a single feature, namely, "*ambiguous_sent_per_sentences*" (i.e. the density of ambiguous sentences, or the number of ambiguous sentences divided by the total number of sentences in a section), at its root. No other feature is included by the tree by C4.5 algorithm affirming that the feature alone was enough for the classification task.

Now, after examining the tree of Figure 26, we could have built the discourse classifier using only one feature, i.e. only checking if a passage had one or more ambiguous sentences; and if did, then we could have classified the passage as "Ambiguous", and otherwise, "Unambiguous". Although this would have simplified the processing steps of the classifier reducing the execution time and space requirements of the program, we chose not to hard-code this rule into our classifier. Otherwise, it would have been to endorse this decision-tree as the only static tree required to detect ambiguity, which it may not be the case real life. It would have also ignored the chance of the two other strong discourse features — "***words /sentences***" and "***Uniques /word***", which are, however, useful features in discourse (for example, "***words /sentences***" is used as readability measure of documents in [Wilson, 1997] and [Gnesi et al., 2005], "***Uniques /word***" as a feature of ambiguity in [Wasson et al., 2005] and as a feature of subjectivity in [Wiebe et al., 2004]) to be included in this decision-tree later on by the learner as new training data becomes available in future. Thus, we embed the C4.5 decision tree learner with our discourse classifier to keep the learning process dynamic,

and to check the applicability of the other comparatively less strong features in classification every time, when learning from new training data.

We used our classifier to carry out the following experiment.

6.3.4 Experiment and Results

In this experiment, our objective was to test the accuracy of using the discourse-level classifier presented in section 6.3.3 in detecting ambiguity. The experiment considered three features, the new feature, "***ambiguous-sentences /sentence***" (*ambiguous_sent_per_sentence*), along with "***Uniques /word***" (*uniques_per_word*) and "***words /sentences***" (*words_per_sentence*). As discussed in section 6.3.1, we used the pre-processed discourse-level corpus (of size 165) for the experiment, and considered the three features mentioned above. The batch program, "*ishrarExperimentC.bat*", was used to run the experiment and display the result on-screen. Like before, the program uses the previously saved feature values and trains the java implementation of C4.5 (revision 8) algorithm. It then tests the classifier once using the same corpus, which it is trained on, and the other time, using 10-fold-cross-validation technique. The results are shown in Table 21.

	Scheme	Correctly Classified Instances	Incorrectly Classified Instances	Kappa	Remarks on the decision tree
Newly Preprocessed Discourse-level Corpus Set (Size = 165)	Training + Testing on same corpus set	146 (88.48%)	19 (11.51%)	0.7572	The tree is very simple with one feature at the root, see Figure 26 of section 6.3.3
	Cross-validation (10 Folds)	143 (86.67%)	22 (13.33%)	0.7203	

Table 21: Results of using the new discourse-level classifier, presented in section 6.3.3

Table 21 presents better results showing an accuracy of 88.48% when training and testing were done on the same data, and 86.67% when 10-fold-cross-validation was performed. The tree being very simple was also of desirable characteristics (see Figure 26 of section 6.3.3). We also find that the agreement between the system's predicted classifications and the actual

annotations was significantly high (Kappa index 0.7572 and 0.7203, when training and testing were done on the same corpus, and when performing 10-fold-cross-validation respectively).

Thus, this result again affirms the optimum efficiency of our sentence classifier in classifying most of the sentences correctly in terms of ambiguity in surface understanding. Using the new feature, density of ambiguous sentences in an instance, derived by the sentence classifier, radically improved the overall efficiency of our new discourse-level classifier, clearly placing it above the classifiers used in our previous experiments, discussed in section 6.2.6.

In the next section, we analyze the performance of our classifiers in terms of their applicability in a practical environment.

6.4 Performance Evaluation Compared to Human

Performance of a system can be evaluated from different perspectives. For evaluating the performance of our system we first chose to compare its performance with the level of agreement human annotators displayed while annotating our discourse-level corpus.

6.4.1 Discourse Classifier That Used Discourse Features

Measuring the inter-annotator agreement of our annotators in annotating passages from our discourse-level corpus showed how efficiently human with *average* knowledge in requirements analysis can detect ambiguity in terms of surface understanding. The results of this measurement in kappa, explained in section 4.4.2.2, pointed out that in detecting ambiguity at the level of surface understanding the average pair-wise agreement of the four annotators with the gold-standard (when decided by the median of the annotators' scores) is **0.6661** in terms of the kappa index (see section 4.4.2.2 to find the agreement measure for each of the annotators individually). We found this value of kappa to be the indication that our human annotators had "substantial" level of agreement (see Table 2 of section 2.7) and the process of detecting ambiguity can be automated.

Now, for all the experiments in section 6.2.6, the gold standard annotations of the discourse-level corpus were similarly decided by the median of the annotators' discourse-level scores, reflecting their collective decision on the annotation of each instance. We, therefore, compare the performances of the discourse-level classifiers that use discourse features, used in all those experiments, opposed to that of the human annotators, by measuring their level of agreement with the gold standard in terms of kappa index values, as shown in the following figure:

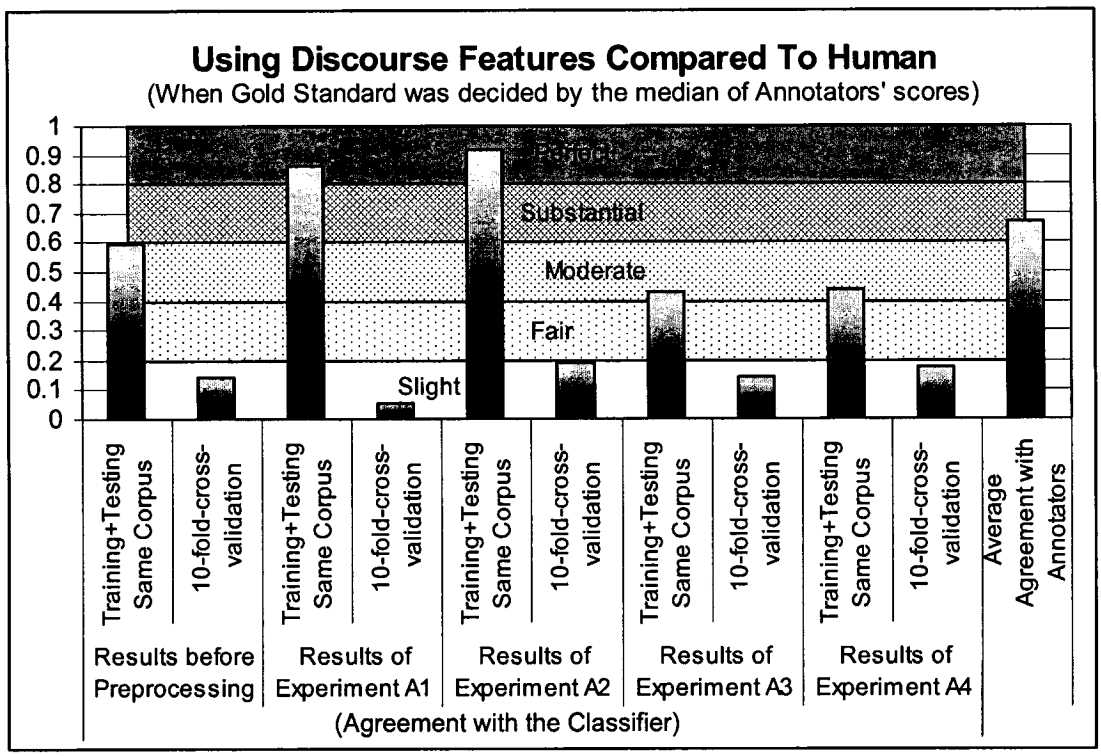

Figure 27: Performance evaluation of discourse-level classifier that uses discourse features compared to human annotators in terms of their level of agreement with the gold-standard, which were decided by the median of the annotators' scores

Figure 27 shows that experiments A1 and A2 over-fitted the training data with hugely sparse trees, which resulted in exceptionally higher degree of agreement with the gold-standard when training and testing were done on same corpus, but very poor agreement when 10-fold-cross-validation was performed. Again, simplification of the tree by combining features, as done in experiments A3 and A4, resulted in even lower agreement with the gold-standard, all

of which exhibits poor performance comparing to that of human annotators. This re-establishes the fact that the discourse-level classifier that use discourse features is unusable in practical fields.

Results of these experiments are described in details in section 6.2.6, whereas the reasons of this poor performance of the discourse-level classifier are thoroughly investigated in section 6.2.7.

6.4.2 Discourse Classifier That Used Sentence Classifier

Our sentence classifier was trained with the sentence-level annotations of *Annotator3* (reviewed and refined by *Annotator1*), and our discourse-level classifier that uses our sentence classifier was trained with our discourse-level corpus, where the gold standard of each instance was, therefore, decided exclusively by the discourse-level score of *Annotator3* only.

Thus, to compare the performance of our discourse-level classifier to the human performance, we compare the level of agreement of the classifier and that of each of the annotators with the gold standard decided by the scores of Annotator3. Figure 28 shows the comparison:

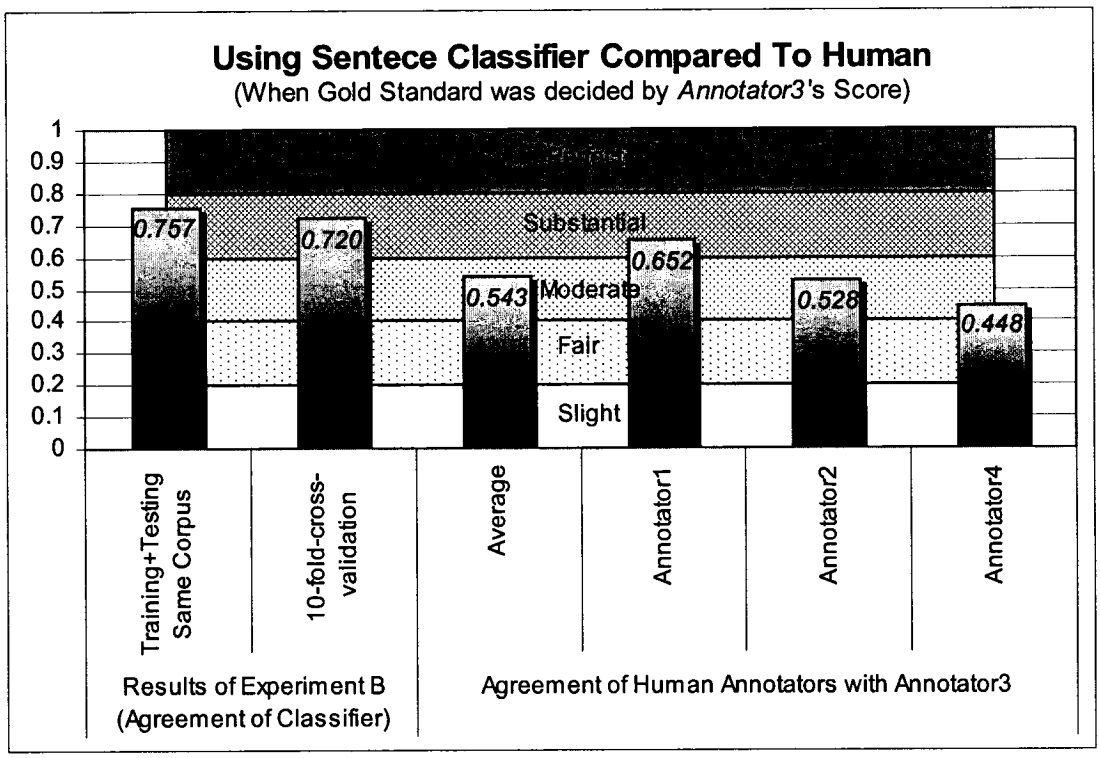

Figure 28: Performance evaluation of discourse-level classifier that uses the sentence classifier compared to human annotators in terms of their level of agreement with the gold-standard, which were decided by the scores of *Annotator3*

Figure 28 shows that the classifier emulates the annotations of *Annotaor3* successfully by fitting its decision tree properly with the annotator's annotation and attaining high "substantial" level of agreement when training and testing were done on the same corpus, and also when 10-fold-cross-validation was performed (kappa index 0.7572 and 0.7203, respectively). This result is also considerably higher than the level of agreement human annotators had with *Annotator3* on average (Kappa index 0.5426 on average, denoting a "moderate" level of agreement). Now, it should be mentioned that if we could expose our annotators to extensive and repetitive training sessions on requirements analysis over a longer period of time (as it was done in the studies of [Wiebe et al., 2004]), other annotators' agreement with *Annotator3* would have improved (for details, see chapter 8, where we discuss the limitations of our project). Among our four annotators, *Annotator3*, a professor in Software Requirements Specification, was the only expert we could avail for our corpus

annotation project. The other three annotators were all from the fields of computer science and software engineering, and familiar with the concepts of requirements analysis.

Thus, we can affirm that our group of annotators, altogether, in general, represented human with *average* knowledge in requirements analysis. And, the level of their average agreement with *Annotator3*, therefore, portrays the performance of a human with average knowledge in requirements analysis compared to that of an expert. Our classifier, being able to outperform such a human, proves its credibility to be applied in practical fields of requirements analysis.

As the discourse-level classifier system aims to be a part of the requirements elicitation and analysis process, where its outputs are expected to be verified by an experienced requirements analyst, we can endorse its performance to be fully adequate for the purpose.

6.5 Compared To Existing Tools or Approaches

Although we intended to compare performances of our classifier with the tools that were used in the related studies, mentioned in chapter 3, we had no access to their systems to gather the required data to compare their runtime performances and/or their accuracies.

However, Table 22 presents a comparison of different features between our classifier and the tools/approaches used in other studies for detecting ambiguity in software requirements specification.

Topics of Comparison	**Our Classifier**	**Manual Detection** by [Kamsties et al., 2001] and [Meyer, 1985]	**Circe** by [Ambriola & Gervasi, 1997]	**QuARS** by [Fabbrini et al., 2001]	**ARM** by [Wilson et al., *1996]*	**Restricting NL** by [Cyre, 1995], [Denger et al., 2003], [Fantechi et al., 1994], [Heinrich et al., 1999], [Lu et al., 1995], [Rolland and Proix, 1992] and [Tjong et al., 2006]	**Newspeak** by [Osborne and MacNish, 1996]
Human Involvement	Required only once, when annotating the training corpus	Required for all steps	Detection and Validation done by human, after deriving models automatically	Required to build the dictionary of ambiguous keywords for each problem domain	Not required	Not required	Not Required
	All the systems/approaches require manual validation of their final outputs						
Able to deal with Unrestricted NL Text ?	Yes	Yes	Partially, text is simplified by human beforehand	Yes	Yes	No	Yes
Dynamic Generation of Ambiguous Keywords ?	Yes, based on training corpus	No, fixed set of keywords in mind varies with the expertise of an analyst	No, ignores the existence of ambiguous keywords	No, manually listed in the dictionaries	No, hard-coded list in the system	No, inherently free of ambiguity	No, ignores the existence of ambiguous keywords
Dynamic Generation of Rules to Detect Ambiguity ?	Yes, based on training corpus	No, uses a fixed set of hand-written rules	No, a fixed set of rules hard-coded in the system	No, a fixed set of rules hard-coded in the system	Rules not available	No, uses a fixed set of rules to write unambiguous language	Not required by the architecture
Attempts to detect ambiguity at the level of "Surface understanding"	Yes	Yes	No	Yes	Yes	No	Yes
Attempts to detect ambiguity at the level of "Conceptual understanding"	No	Yes, by manual formalization	Yes, with the aid of human	No	No	Yes, with the aid of human and/or by theorem provers of formal logic	No

Table 22: Comparison of our classifier with the systems/approaches used in with studies

6.6 Conclusion

All these results of performance thoroughly prove our hypothesis affirming that the approach of using a text classifier is applicable in the practical fields for detecting ambiguous passages in SRS documents. Although our work is fully concentrated in detecting ambiguities at the level of surface understanding, it will facilitate our future work of detecting conceptual ambiguities by improving the quality of the SRS text.

Chapter 7

Design and Implementation

7.1 Domain Model

The domain model of our system is shown in Figure 29.

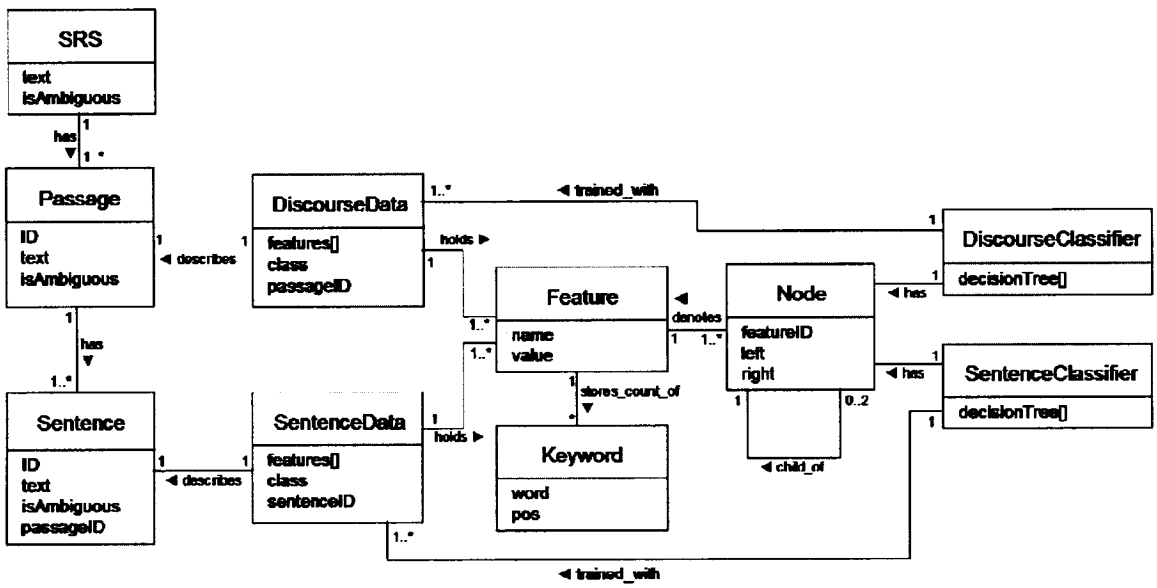

Figure 29: Domain Model of our system

7.2 The Prototype

We named the prototype of our system, Requirements Specification Ambiguity Checker (ReqSAC). The system can label a discourse as ambiguous, and, at the same time, identify the sentences that are responsible for making it so. The UI also features on-the-fly editing of the text to correct the errors and enhance the quality of the SRS, using the same linguistic features that our annotators used to detect errors at the level of surface understanding. As explained in chapters 5 and 6, the *Stanford Parser* [Klein & Manning, 2003] and the *Weka* machine learning workbench [Witten & Frank, 2005] are both embedded in ReqSAC. It is implemented to run both as a standalone application and from within Rational XDE environment. Figure 30 shows a snapshot of its main window (see Appendix III for more snapshots of ReqSAC).

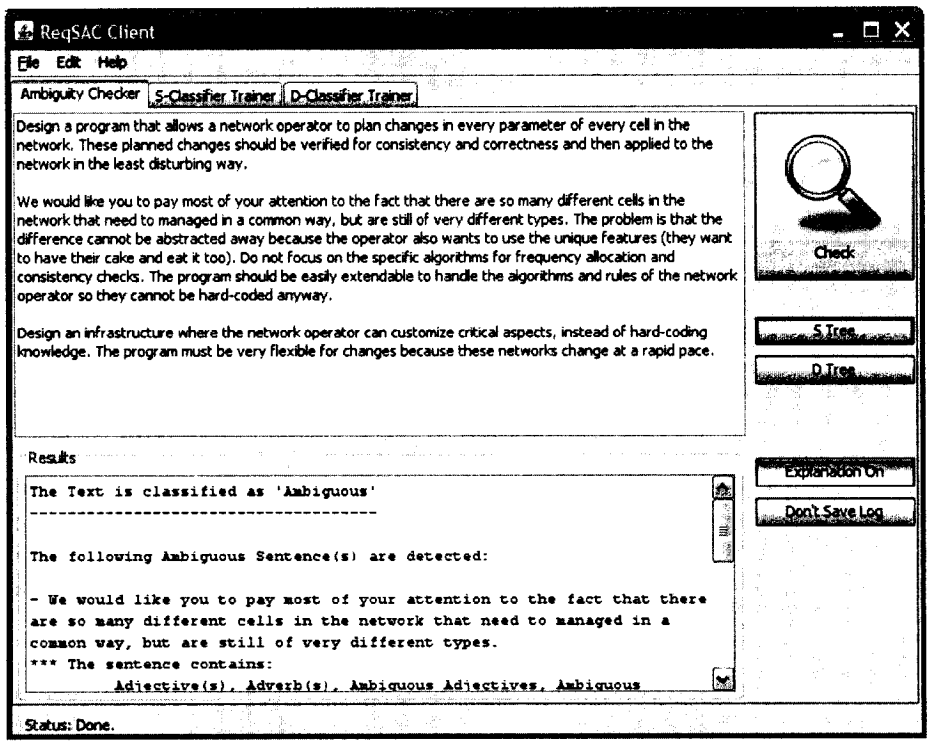

Figure 30: ReqSAC: The prototype of our system

Chapter 8

Discussion

8.1 Introduction

The outcomes of our work have important significance in context of RE. The milestones we achieved in our study did not only prove our concept for practical use, but also introduced some profound analyses that can fuel future researches in this area. We will summarize the scope of value additions of our work in this chapter.

In our work, we also faced many challenges that we had to overcome. We often had to enrich our methodology with new ideas and modifications to endure these obstacles, and thus, satisfied all our objectives. Yet, our accomplishments are not free from limitations. In this chapter, we will also discuss all the limitations we faced that, although, did not hinder our progress in any way, but should be stated here for the correct evaluation of our work.

8.2 Significance of Our Work

Our research objective, as a "proof of concept", was to build a text classification system and test its applicability for detecting ambiguity in SRS documents. The following is the list of outcomes we achieved along the way of completing our objective that are of high significance in the RE field:

(1) We introduced a hierarchical quality model for the decomposition of software requirements text quality into measurable features that would be collected directly from the text (see Figure 2 of section 1.6). The quality model targets the automatic assessment of textual requirements in terms of their ambiguity. The model alone can be used by (semi-) automatic processes of quality assessment of SRS documents.

(2) We built two sets of annotated corpora: one discourse-level corpus (holding 165 samples of passages from 25 problem descriptions, each of different domains), and one sentence-level corpus (holding samples of 1211 different sentences from the same problem descriptions). All the passages and sentences of the corpora are annotated into two distinct categories — "ambiguous" and "unambiguous", using a standard procedure of scoring and employing median voter model [Congleton, 2004]. The corpora can be used for the purpose of training and/or testing future text classifiers in this field. It should be mentioned that to our knowledge there exists no standard annotated corpus in this field.

(3) The analysis on our work of corpus annotation and the level of the annotators' agreement ascertained an important notion that the eventual extraction of the features can lead to successful classification of ambiguity in both surface and conceptual understanding. However, for this purpose, more improvement in NLP tools in terms of accuracy is essential, especially for deeper semantic analysis required by conceptual understanding. The analysis also showed that the task of detecting ambiguity is relatively difficult for human. Moreover, it indicated a positive correlation between the surface and conceptual understanding of the text, and a negative correlation between the understanding and the time required to analyze a text. The above, therefore, confirms that lowering the level of surface ambiguity would lead to a better conceptual understanding of the requirements and reduce the time needed for requirements analysis.

(4) We have been successfully devised an NLP-driven feature extractor tools that extract discriminating features of the SRS text influencing ambiguity in surface

understanding. The tools can output data files (in standard ARFF format), each representing an annotated corpus in terms of feature vectors. The file can be used to train any machine learning algorithm to develop a classifier.

(5) We finally developed a decision tree-based text classifier that works at both sentence and discourse levels of an SRS document and detects ambiguity in surface understanding. The classifier can also be trained with new training data, which makes it robust to deal with unseen samples of SRS in future. The classifier attained an impressive accuracy, high enough to be applicable in a practical semi-automated environment. This eventually proves our hypothesis of the usability of a text classifier in detecting ambiguity in SRS documents.

(6) We also developed a small prototype (written in Java) to demonstrate the use of our classifier. The prototype is platform independent, and can be executed online, or from within Rational XDE environment. This ensures future usability of the system.

8.3 List of Limitations

The following is a list of the limitation we encountered during our work:

(1) We could avail limited time to conduct the series of long training of our human annotators. We selected annotators who are already familiar with detecting ambiguity in software requirements specification and carried out a brief training session with a pilot study followed by sessions of groups discussions (section 4.4.1 presents the details). Yet, we feel multiple training sessions over a longer period of time could have yielded a better agreement among the annotators.

(2) Limited training sessions compelled us to find annotators, who were already familiar with detecting ambiguity in software requirements specification. This led us to find four annotators for discourse-level annotation, and only one annotator for sentence-level annotation (the sentence-level annotations were later

reviewed and refined by two annotators). They are, although, not many in number, but, considering a lot of other notable researches (e.g. [Wiebe et al., 2004], which used a corpus annotated by one annotator only, and [Ide et al., 2002; Poesio & Vieira 1998], which used corpora annotated by two annotators), we find that their number is, however, in the acceptable range. Yet, having a large number of trained annotators may strengthen the appeal of our annotations even more.

(3) To our knowledge, there exists no standard annotated corpus in this field. Thus, we had to build our own annotated corpus, which eventually limited the size of our corpus for training and testing. As our classifiers use decision tree learning algorithm, with the inclusion of new training data, the system will be more efficient in future.

(4) Our sentence-level feature extractor tool has a built-in keyword generator that dynamically generates lists of ambiguous keywords of different parts of speech. By default, each of these keywords is one word. To avoid complexity and reduce development time, we ignored the existence of possible multi-word terms as an ambiguous keyword. An example is "a lot of". The words "a", "lot" or "of", individually, are not of ambiguous characteristics. Thus, our keyword generator, based on our training data, should never list any of these words in the list of ambiguous keywords that it automatically generates. But, "a lot of" altogether as a collocation is found to be of ambiguous characteristics in the following sentence from our training data:

"*...the thermostat has a lot of work to do to keep the temperature even...*"

8.4 Conclusion

The above limitations only suggest the possibility of valuable additions to our work. We, therefore, intend to deal with these issues as our future work in this project. Now, despite

these limitations, we found that our work provided a significant contribution in the RE field and proved our concept and established the idea of using our new approach successfully in detecting ambiguity from SRS documents.

Chapter 9

Conclusion and Future Work

This thesis addressed the problem of detecting ambiguities in SRS documents. Acknowledging the fact that none of the previous work has tested the applicability or performance of using a text classification system to automate such detection process, this research proved our hypothesis by affirming that the approach of using a text classifier is applicable for detecting ambiguous passages in SRS documents. The work also encompassed some related important topics, e.g. how difficult it is to detect ambiguity manually from SRS documents and how the automatic tools developed can compare to human performance.

To prove our concept, we developed a text classification system that can detect ambiguity in an SRS document by classifying its passages as ambiguous or unambiguous. The system yielded high accuracy in performance demonstrating results with 86.67% accuracy using 10-fold-cross-validation technique. Comparing its results of its degree of agreement with the decisions of an expert, it outperformed human annotators with *average* expertise in detecting ambiguities. It can also be affirmed that the system will perform better in practical fields with the inclusion of new training data. We also built a prototype of this system to demonstrate its use.

Our work concentrated in detecting surface understanding ambiguities only, avoiding the detection of ambiguities, which are at the level of conceptual understanding. The reason for this was the unavailability of efficient resources till this date that are able to perform semantic analysis or hold domain knowledge, required for detecting the presence of the features in text that degrades conceptual understanding of an SRS document. This also re-affirmed the claims of Meyer [1985] and Kamsties *et al* [2001] that ambiguities of SRS

documents may not be detected without the aid of formalism. Thus, we proposed that our NLP-driven quality assessment project would deal with detecting ambiguity at the level of conceptual understanding by introducing a level of formalism (with domain model generation and simulating the path of execution) and following a semi-automated approach with continual feedback of the client (see Figure 1 of chapter 1). But, this thesis, being a part of the project, successfully establishes its approach of detecting the ambiguities that are at the level of surface understanding, and, thus, facilitating the processes of semi-automated analysis, to be done by the upcoming modules. We strongly believe our system, with the potential to clean up ambiguities at the level of surface understanding, will not only serve the aforesaid project, but also be useful as a standalone application working in conjunction with the requirements specification writing tools. The prototype of this system is, therefore, implemented to run both as a standalone application and from within Rational XDE environment.

Although our system with all the results of its performance proves our concept and establishes the idea of using our new approach successfully in detecting ambiguity from SRS documents, our future work should still focus on introducing more training data to improve its efficiency in dealing with unseen SRS texts. As mentioned in chapter 7 on discussing our limitations, we would suggest including a collocation extractor with our keyword generation tool, which will improve its efficiency even more. Finally, we look forward to its successful integration with the module that detects ambiguity in conceptual understanding.

Bibliography

[Ambriola & Gervasi, 1997] Ambriola, V., & Gervasi, V. (1997). "Processing natural language requirements". In *Proceedings of Automated Software Engineering (ASE'97): 12th IEEE International Conference*, November 1–5, 1997, 36–45.

[Best, 1995] Best, L. (1995). Delinquency Collections. OOPSLA DesignFest®. Last retrieved on July 18, 2007 from http://designfest.acm.org/Problems/Delinquency/Delinquency_95.htm

[Brill, 1992] Brill, E. (1992). "A simple rule-based part-of-speech tagger". In *Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP-92)*, Trenton, Italy, April 1–3, 152–155.

[Carletta, 1996] Carletta, J. (1996). "Assessing agreement on classification tasks: The kappa statistic". *Computational Linguistics, 22(2)*, 249–254.

[Chomsky, 1965] Chomsky, N. (1965) *Aspects of the theory of syntax*. MIT Press.

[Cohen, 1960] Cohen, J. (1960). "A coefficient of agreement for nominal scales". *Educational and Psychological Measurement, 20*, 37–46.

[Congleton, 2004] Congleton, R.D. (2004). "The Median Voter Model". *Encylopedia of Public Choice: Volume II* (eds. Charles. K. Rowley and Friedrich Schneider), ISBN: 0-7923-8607-8, 382–387.

[Cyre, 1995] Cyre, W. R. (1995) "A Requirements Sublanguage for Automated Analysis". *International Journal of Intelligent Systems, 10 (7)*, July 1995, 665–689.

[Dasarathy, 1991] Dasarathy, B.V. (ed.). (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, ISBN 0-8186-8930-7.

[Denger et al., 2003] Denger, C., Berry, D., & Kamsties, E. (2003) "Higher Quality Requirements Specifications through Natural Language Patterns". In *Proceedings of SWSTE, IEEE International Conference on Software-Science, Technology & Engineering*, 2003, p. 80.

[IEEE, 1998] The Institute of Electrical and Electronics Engineers, Inc. (1998). *IEEE recommended practice for software requirements specifications* (IEEE Std 830-1998), 20 October, 1998, ISBN 0-7381-0332-2, New York: Author.

[Fabbrini et al., 2001] Fabbrini, F., Fusani, M., Gnesi, S., & Lami, G. (2001). "An Automatic Quality Evaluation for Natural Language Requirements". In *Proceedings of the Seventh International Workshop on Requirements Engineering: Foundation for Software Quality REFSQ'01*, Interlaken, Switzerland, June 4–5, 2001.

[Fantechi et al., 1994] Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., & Moreschini, P. (1994) "Assisting requirement formalization by means of natural language translation". *Formal Methods in System Design, vol. 4*, 243–263.

[Fleiss, 1971] Fleiss, J.L. (1971). "Measuring nominal scale agreement among many raters". *Psychological Bulletin, 76*, 378–382.

[Harris, 1998] Harris, G., (1998). Case Management: A Framework. OOPSLA DesignFest®. Last retrieved on July 18, 2007 from http://designfest.acm.org/Problems/CaseManagement/CaseManagement_98.htm

[Heinrich et al., 1999] Heinrich E., Kemp E., & Patrick J.D. (1999). "A Natural Language Like Description Language". In *Proceedings of the 10th Australasian Conference on Information Systems (ACIS)*, 375–386.

[Heliotis et al., 2003] Heliotis, J., Freeman-Benson, B., & Paisley, B. (2003). Data Collection in the Forest. OOPSLA DesignFest®. Last retrieved on July 18, 2007 from http://designfest.acm.org/Problems/DataCollection/DataCollection_03.htm

[Ide et al., 2002] Ide, N., Erjavec, T., & Tufis, D. (2002) "Sense discrimination with parallel corpora". In *Proceedings of the ACL-02 workshop on Word sense disambiguation: recent successes and future directions*, July 11, 2002, 61–66.

[Just & Carpenter, 1987] Just, M.A., & Carpenter, P.A. (1987). *The psychology of reading and language comprehension*. Boston: Allyn and Bacon.

[Kamsties et al., 2001] Kamsties, E., Berry, D.M., & Paech, B. (2001). "Detecting Ambiguities in Requirements Documents Using Inspections". In *Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01)*, July 23, 2001, Paris, France, 68–80.

[Klein & Manning, 2003] Klein, D., & Manning, C. D. (2003). "Accurate Unlexicalized Parsing". In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, 2003, 423–430.

[Kleinberg, 1999] Kleinberg, J. (1999). "Authoritative sources in hyperlinked environment". *Journal of the ACM, 46(5)*, 604–632.

[Kolcz & Alspector, 2001] Kolcz, A., & Alspector, J. (2001) "SVM-based filtering of e-mail spam with content-specific misclassification costs". In *Proceedings of workshop on Text Mining, IEEE ICDM-2001*, IEEE Press, Piscataway, NJ.

[Kraemer et al., 2004] Kraemer, H. C., Periyakoil, V. S., & Noda, A. (2004). "Kappa coeffcients in medical research". *Tutorials in Biostatistics Volume 1: Statistical Methods in Clinical Studies*, John Wiley & Sons, Ltd., ISBN: 0-470-02365-1.

[Krenn et al., 2004] Krenn, B., Evert, S., & Zinsmeister, H. (2004). "Determining intercoder agreement for a collocation identification task". In *Proceedings of Konvens'04*, September 2004, Vienna, Austria, 89–96.

[Kriens, 1996] Kriens, P. (1996). CellKeeper, a Cellular Network Manager. OOPSLA DesignFest®, 1996. Last retrieved on July 18, 2007 from http://designfest.acm.org/Problems/CellKeeper/CellKeeper_96.htm

[Gnesi et al., 2005] Gnesi, S., Lami, G., Trentanni, G., Fabbrini, F., & Fusani, M. (2005). "An Automatic Tool for the Analysis of Natural Language Requirements". *International Journal of Computer Systems Science and Engineering, Special issue on Automated Tools for Requirements Engineering, 20(1)*, Leicester, UK: CRL Publishing Ltd., January 2005.

[Landis & Koch, 1977] Landis, J.R., & Koch, G.G. (1977). "The measurement of observer agreement for categorical data". *Biometrics, 33*, 159–174.

[Larsen & Aone, 1999] Larsen, B., & Aone, C. (1999). "Fast and effective text mining using linear-time document clustering". In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, August 15–18, 1999, San Diego, California, 16–22.

[Layda, 2000] Layda, T. (2000). Order Matcher for an Electronic Stock Market. OOPSLA DesignFest®. Last retrieved on July 18, 2007 from http://designfest.acm.org/Problems/OrderMatcher/OrderMatcher_00.htm

[Leffingwell & Widrig, 2003] Leffingwell. D., & Widrig, D. (2003). *Managing Software Requirements: A Use Case Approach* (2nd ed.). ISBN 0-321-12247-6, Boston: Addison-Wesley

[Letier et al., 2005] Letier, E., Kramer, J., Magee, J., & Uchitel, S. (2005). "Monitoring and Control in Scenario-Based Requirements Analysis". In *Proceedings ICSE 2005 - 27th International Conference on Software Engineering*, St. Louis, Missouri, USA: ACM Press, May 2005.

[Lin, 1998] Lin, D. (1998). "Dependency-based Evaluation of MINIPAR". In workshop on the Evaluation of Parsing Systems, Granada, Spain, May 1998.

[Lu et al., 1995] Lu, R., Jin, Z., & Wan, R. (1995). "Requirement Specification in Pseudo-Natural Language in PROMIS". In *Proceedings of 19th International Computer Software and Applications Conference (COMPSAC'95)*, 96–101.

[Meyer, 1985] Meyer, B. (1985) "On Formalism in Specifications". *IEEE Software, 2(1)*, January 1985, 6–26.

[Ormandjieva et al., 2007] Ormandjieva, O., Kosseim, L., & Hussain, I. (2007). "Toward Text Classification System for Quality Assessment of Software Requirements Written in Natural Language". Accepted at the SOQUA 2007 workshop.

[Osborne & MacNish, 1996] Osborne, M., & MacNish, C.K. (1996) "Processing natural language software requirement specifications". In *Proceedings of ICRE'96: 2nd IEEE International Conference on Requirements Engineering*, IEEE Press, 229–236.

[Quinlan, 1993] Quinlan, J.R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

[Poesio & Vieira 1998] Poesio, M., & Vieira, R. (1998). "A corpus-based investigation of definite description use". *Computational Linguistics, vol.24 n.2*, June 1998, 183–216.

[Rolland & Proix, 1992] Rolland, C., & Proix, C. (1992). "A Natural Language Approach For Requirements Engineering". In *Proceedings of the Fourth International Conference CAiSE'92 on Advanced Information Systems Engineering, vol. 593 of Lecture Notes in Computer Science*, Manchester, UK, 257–277.

[Rousseeuw & Leroy, 1987] Rousseeuw, P.J., & Leroy, A.M. (1987). *Robust regression and outlier detection*, New York: Wiley.

[Spertus, 1997] Spertus, E. (1997). "Smokey: Automatic recognition of hostile messages". In *Proceedings of the Ninth Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-97)*, Providence, RI, July 27–31, 1058–1065.

[Tjong et al. 2006] Tjong, S.F., Hallam, N., & Hartley, M. (2006). "Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns". In the proceedings of the Sixth IEEE International Conference on Computer and Information Technology, 2006 (CIT '06), September 2006, 199-199

[The Standish Group International, Inc., 1995] The Standish Group International, Inc. (1995). *The CHAOS Report.*

[Wasson et al., 2005] Wasson, K.S., Schmid, K.N., Lutz, R.R., & Knight, J.C. (2005). "Using Occurrence Properties of Defect Reporting Data to Improve Requirements". In *Proceedings of the 13th IEEE International Requirements Engineering Conference (RE'05)*, 29 August - 2 September 2005, Paris, France, 253–262.

[Weiss et al., 2005] Weiss, S.M., Indurkhya, N., Zhang, T., & Damerau F. (2005). *Text mining: Predictive methods for analyzing unstructured information.* New York: Springer.

[Wiebe et al., 2004] Wiebe, J., Wilson, T., Bruce, R., Bell, M., & Martin, M. (2004). "Learning Subjective Language". *Computational Linguistics, v.30 n.3 (September 2004)*, ISBN:0891-2017, Cambridge, MA, USA: MIT Press, 277–308.

[Wiebe et al., 2003] Wiebe, J., Breck, E., Buckley, C., Cardie, C., Davis, P., Fraser, B., Litman, D., Pierce, D., Riloff, E., Wilson, Theresa., Day, D., & Maybury, M. (2003). "Recognizing and organizing opinions expressed in the world press". In *Working Notes of the AAAI Spring Symposium in New Directions in Question Answering*, Palo Alto, CA, 12–19.

[Wilson, 1997] Wilson, W. (1997). "Writing Effective Requirements Specifications". In *USAF Software Technology Conference*, Utah, April 1997.

[Wilson et al., 1996] Wilson, W., Rosenberg, L., & Hyatt, L. (1996) "Automated Quality Analysis of Natural Language Requirement Specifications". In *Proceedings of 14th Annual Pacific Northwest Software Quality Conference*, Portland.

[Witten & Frank, 2005] Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Francisco, CA: Morgan Kaufman.

[Zhang & Varadarajan, 2006] Zhang, Z., & Varadarajan, B. (2006). "Utility scoring of product reviews". In *Proceedings of the 15th ACM international conference on Information and knowledge management*, Arlington, Virginia, USA, 51–57.

Appendix I

Each of the passages in our discourse-level corpus is saved as individual ASCII text files. Their filenames hold the serialized ID numbers of the problem descriptions that they belong to. We present 30 passages from our discourse-level corpus in the following:

1. p01s01.txt

Case Management is a business function common to government health benefit programs, and insurance and financial organizations. The problem you are here to solve today is to develop an object oriented framework for use in developing specific case management systems to support case workers in these industries. These systems must provide access to all necessary information for case workers to process work or respond to customer service needs. They may also provide workflow and business rules processing to support case workers. As the main incoming channel for cases is frequently the telephone, the system must have sub-second response times and be intuitive to use.

---

2. p01s02.txt

Many organizations use Cases as their basic unit of business. Consider, for example, government benefit programs, insurance companies and lending institutions. A government agency, in administering a benefit program, controls and tracks the issuing of cheques and benefits to particular clients according to legislation, policy, rules and regulations, making decisions on a case by case basis. Insurance companies must manage claims against policies, making decisions for each case. Lending institutions grant or refuse loans for a case based on specific criteria and regulations. The common element in each of these is the Case.
A case is a specific instance of applying procedures to render a decision. It involves a client, a specific set of rules in force at the time when the case started, and follows a lifecycle. Case Management involves managing client tombstone data, supporting the workflow specified by the case lifecycle, managing events and case status, and generating bring-forward actions. These are described in more detail below. In addition, each specific case management application will have analysis, display, and processing requirements unique to the company, which, in the interests of simplicity, are not part of this problem.

Common functions for case management include:

Capture and view tombstone information about an applicant, client, or case
Link the cases to other files and/or documentation (including integration with imaging systems)
Prioritise the cases based on specific rules (including Artificial Intelligence engines that deal with complex rules)
Manage the case history, update status, log relevant activities and events (e.g., receipt of documentation, dispatch of correspondence, scheduling of a meeting)
Assign the cases to appropriate personnel, manage work, track performance
Capture and process business specific information
Create and track Bring-Forward items
Schedule work and associated resources (e.g., booking courtrooms and judges)

Prepare and print reports

The design problem for DesignFest® is to create an Object Oriented framework for use in developing Case Management applications quickly, and customisable to the particular work practices of the company purchasing the framework.

Four areas, correspondence management, contact tracking, prioritising cases based on rules, and scheduling work and resources, have been excluded from the scope of the framework to simplify the problem.

---

3. p01s03.txt

Client data and case tombstone data management refers to the capture and tracking of basic client data such as name, addresses, and phone numbers. It also includes capturing and tracking similar data for clients representatives, and the nature of the relationship.

This process also includes capturing data unique to the case management application under development. The framework must provide facilities for defining data that needs to be captured. There is no requirement for this to be configurable at run time (but it wouldn hurt if that was an option).

---

4. p01s04.txt

Workflow support refers to the part of the system that allows multiple workers to work on multiple cases in a wide variety of organisational arrangements.
The framework must support workflow functions that are managed either by assigning cases to individual case workers or to a pool of case workers of the same subtype. The use-case descriptions below will specify when this applies. In the former situation, the cases are assigned centrally by a line manager. In the latter situation, the organisation using the framework for developing their case management system has essentially chosen to empower their case workers to partially manage their workload. These case workers will pick a case from the pool to work on next. The manager is depending on their professionalism to ensure that cases do not stay in the pool too long.

At a minimum, the framework should support:

A case worker is assigned cases which are managed from start to finish.
A case worker is assigned a case by a manager to perform a specific function for that case, and then informs the manager the task is complete. The manager then determines the next appropriate step (close, assign to another case worker for the next processing step, put aside until a bring-forward is activated, etc.).
A case worker selects cases from a pool which are managed from start to finish.
A case worker selects cases from a pool to perform a specific function, and then moves that case to the next appropriate step (close, put in another pool, put aside until a bring-forward is activated, etc.).
Workflow support for case workers and managers requires an in-basket metaphor for selecting one of the cases assigned or selecting a case from a pool. A manager should be able to view, control and organise the work among the case workers.
Workflow support also includes integration with e-mail and other means of communication in an office environment.

Finally, workflow support also means providing the ability to Approve or Reject a particular request regarding a case. A case may involve a single or multiple requests requiring approval, and security may be needed to restrict who can make these decisions, depending on the organisation building the application using this framework.

---

5. p01s05.txt

Event and status management forms the core of the case management framework. The sequence of events trace the processing of the case by the organisation. These are referred to frequently, and may ultimately be used to resolve disputes, respond to queries, and

predict trends in the business. The current status of a case situates the case within the lifecycle model described by the organisation procedures, and may be used to determine the next step in processing.
Event management refers to the creation, editing and reporting of events significant to the business. The framework must provide facilities to configure, at run time, the types of events and the information to capture for each event. An event may trigger a status change, or close a bring-forward.

Status management refers to maintaining a status field for the case. The status values may control other aspects of the framework, depending on the configuration. For example, the framework must support restricting the next status or the types of events that may occur based on the current status value. Changing to a particular status may cause an event to be created automatically, such as creating a bring-forward entry due at some future date. Changing to a particular status may also automatically close a bring-forward (whether it is due or not).

---

6. p01s06.txt

Bring-Forward entries are reminders to a case worker that it is time to perform a certain action. The framework must provide a means to manually and automatically create, review, and close bring-forwards.

---

7. p01s07.txt

The framework must be scalable for several hundred case workers and tens of thousands of cases and clients. Case workers may be centrally located, or geographically dispersed. If case worker are geographically dispersed, then you may assume that the cases are managed by geographic regions.
Response times for case workers must be less than two seconds on saving and retrieving case information, and sub-second on moving between views of different data for the current case.

Line Management requires daily and weekly reporting. Senior Management requires the ability to regularly coalesce case data from disperse locations for data warehousing and data mining in support of Decision Support Systems. Usually this will be weekly, monthly and quarterly.

Security is needed to control which users can do certain actions, such as assigning cases, approving a request, or editing particular fields.

The system must support multi-lingual deployment using multi-byte character encoding.

The system must include an audit trail for each case.

The system must include archiving of old or inactive data.

---

8. p02s01.txt

A Flexible Manufacturing System (FMS) is a computerized factory production entity, composed of numerically controlled machines, an automatic inventory system for storing raw, temporary in-process, and finished pieces, and an automatic guided vehicle system (AGV) for moving pieces, which can operate on different types of pieces.

---

9. p02s02.txt

The production is planned at factory level and is represented by lots assigned to the FMS (a lot is an administrative entity indicating a group of identical pieces to be manufactured together.) Several lots of different types of pieces can be in production at the same time in a given FMS.

Each type of piece requires a specific ordered sequence of operations. A machine can perform one or more of those operations, but only a single operation on one piece at a time. Each operation takes a certain amount of time.

---

10. p02s03.txt

A network connects the FMS computer to the machine, inventory, and AGV controllers. The real-time dispatcher of the FMS assigns pieces to the machines as soon as they become idle, and issues requests of missions to the AGV system, monitoring the shop-floor.

Pieces are taken one at a time from the input store and transported to an idle machine where the first operation is performed. As soon as the operation is finished, the piece is evacuated from the machine and transported elsewhere; either to another machine for performing the next operation or to one of the temporary stores. Machines are assigned to a waiting piece (if any) as soon as they become idle. A machine is idle when it is not performing an operation on a piece and the last piece operated on has been evacuated.

The cell control interacts with the AGV through requests of the type "pick up piece X from A and move it to B", where A and B can be one of the stores or the machines. For our purposes (as is the case in a small FMS), the time needed to serve a transport request is small relative to the time needed for a machine operation. When the destination is a machine, the transport request is not made until the machine is idle and the piece has been assigned to it.

---

11. p02s04.txt

As an example, we describe a simple FMS for manufacturing cylindrical mechanical parts, consisting of :

Two turning machines, M1 and M2, and a drilling machine, M3, each of which can work on a single piece at a time. The machines can operate in parallel.
There are two different turning operations, TO1 and TO2, and a drilling operation, DO1.
M1 and M2 can both perform the operations TO1 and TO2; M3 can perform the operation DO1.
The two turning operations take 30 minutes to complete; the drilling operation takes 10 minutes.
Two types of pieces, TP1 and TP2. TP1 requires the operation sequence <TO1, DO1>, and TP2 requires the sequence <TO2, DO1, TO1>.
The store for raw pieces: Store_in.
The store for finished pieces: Store_out.
The store for temporary in-process pieces: Store_tmp. It can hold up to 50 pieces, each of which can be recovered individually from their place in the store.
The transportation system with three carts, each of which can carry a single piece and is equipped with a robot arm that loads or unloads the cart on command from the FMS computer.
The transportation system connects all machines and stores to each other.
There are two different lots, the first consisting of 10 pieces of type TP1, the other of 5 pieces of type TP1 and 5 pieces of type TP2. The lots are processed simultaneously by the FMS.
The controller/dispatcher knows at all times the state of all of the machines, the places in the stores and the carts (free or occupied, and if occupied, by which piece) as well as the position and state of all the individual pieces. Before the production run, the controller is informed of the sequence of operations required by each type of piece, and the configured operations for each machine.

The dispatcher calculates the next operation to be performed for each piece and on which machine, requests that the transportation system move the piece to its next destination, synchronizing the loading and unloading operation of the carts.

---

12. p02s05.txt

When the Drilling Machine M3 has finished the operation DO1 on a piece, it notifies the Dispatcher. The Dispatcher makes a request for a cart from the Transportation System for evacuating the piece from the Drilling Machine.

When the cart has evacuated the piece, the machine enters the IDLE state. The Dispatcher then checks if the piece is finished.

If the piece is finished, it is transported to Store_out; otherwise, the Dispatcher checks if there is an idle machine capable of performing the next operation on the piece.

If an idle machine is found, the piece is transported there; otherwise, the piece is moved to the temporary store Store_tmp.

---

13. p02s06.txt

When a piece is evacuated from the drilling machine M3, the machine becomes idle and it is ready to perform an operation on another piece.

When the machine enters the IDLE state, the dispatcher is informed and checks whether a piece (in the store for raw pieces Store_in or in the temporary in-process pieces Store_tmp) is waiting for the next operation on the drilling machine M3.

When the dispatcher find a piece Pw waiting for an operation, it requests the Transportation System for a cart; the Transportation System is then assigned the task of transporting the waiting piece Pw to the drilling machine.

When the Transportation System has accepted the transportation, the drilling machine exits the IDLE state.

---

14. p02s07.txt

The specification can be adjusted for different levels of complexity applying one or more of the following constraints:

All pieces are identical
Each piece type requires only one operation
Each machine can perform only one type of operation
The machines can break and can be repaired
The transportation system has infinite carts
The transportation system has a limited number of carts
The transportation system has only one cart

---

15. p03s01.txt

Cellular telephone networks have shown exponential growth rates in the past few years. From networks containing a handful of cells to cover a country, they have grown into networks containing tens of thousands of cells. It used to be possible to configure these networks by hand, but the tremendous growth has made this infeasible. There are just too many cells to be handled by man.

We would like you to design a program, called CellKeeper, that can plan and execute changes in the configuration of a live network without disturbing it. This is daunting because of the volume of information involved, the myriad of hardware, and the complex constraints that exists in cellular telephone networks.

---

16. p03s02.txt

A cell in a cellular telephone network handles all the calls of mobile phones in the area it covers. To do this, it transmits identifying information on a beacon radio frequency and handles the "traffic" on other frequencies. When a network is initially built, the area that is covered by each cell is very large. When the number of subscribers increases, capacity is added by splitting cells. This results in more and smaller cells, offering more total capacity. This technique makes cellular networks highly scalable. In busy areas like cities, cell diameters are measured in metres, in rural areas the diameters can be up to 64 km. The only limit is the interference of cells that reuse the same frequency. Networks that use digital signals instead of analog between the cell and the mobile phone have an advantage because the interference levels can be considerable higher at the same quality.

(IMAGE)

One of the most complicated aspects of a cellular network is the planning of the frequencies. This is almost a black art and is usually handled by special planning programs. These programs contain prediction algorithms that take into account the terrain that the cell covers.

---

17. p03s03.txt

The mobile phone must be slaved to a cell when it is first turned on. It will select the most suitable cell by scanning all beacon frequencies. It will then send an identifying signal to the cell to make contact. After this contact is made, the cell is in full charge of the mobile. The only initiative left to the mobile is to broadcast that it wants to start a call. When the mobile is moving, the cell will detect that the signal quality will become worse, and will hand over the mobile to a more suitable neighboring cell.

---

18. p03s04.txt

A cell is implemented by a base station. A base station is a computer plus radio equipment that is connected to a telephone exchange. Base stations are located on a "site". The sites are the most visible part of a cellular network because of the very obvious antenna masts that are visible all over the country. One site can host multiple base stations. The base stations are connected to a telephone exchange that, besides switching the calls, is also responsible for controlling the operation of the cells. In the GSM system it is called the Base Station Controller (BSC).

(IMAGE)

---

19. p03s05.txt

The network operator wants to process as many calls as possible for the least amount of cost. The tasks to achieve this are manifold. Planning, subscriptions, procurement, installation, handling of alarms, measurements, configuration changes and many more. The networks have become so large that they are normally split sections. Each section is then managed by an Operational Maintenance Center (OMC). Each OMC manages a number of BSCs and a geographic area.

For this problem we focus on how to change the network configuration. Configuring the network is done by sending management commands to the cells that set values on parameters or read these parameter values. This is complicated because each cell has literally hundreds of parameters. A short summary:

Naming and identity. E.g. a cell has a "user friendly" name and a unique global identifier
The set of frequencies the cell may use
Output antenna power settings

Relationship to neighbouring cells. E.g. to which cells and when should a handover take place between two cells.
Handover strategy, under what conditions can a handover take place
Access rights from mobile phones. Calls can only allow emergency calls or a certain class of mobile telephones.
Mobile phone parameters. The cell fully controls the operation of the mobile phone and can set many of its operating parameters.
Connection to the available hardware
Many, many others
Several of these parameters are dependent on, and constrained by, the settings of other parameters in other cells. For example, neighboring cells should always use different frequencies. Also, each cell needs to know what its neighbors are. Not only for the cells of the same network, but also for cells of adjacent networks when they have cells that could interfere.

All these parameters for ten thousands of cells are difficult to manage. Many parameters must be changed when cells are split. Not only for the split cells, but also for adjacent cells.

Changing parameters in the network must be done in a short time on all related cells to keep the disturbance to the network as low as possible. For example, changing the frequency plan of a set of cells makes the cells inoperative during the changes. Therefore, the changes must be planned and executed in a short time, usually during the middle of the night. Such a change should of course be done carefully to keep the network consistent.

A very complicating factor in the management of the cells is that in one network, one can find cells from different manufacturers and different versions and variants from the same manufacturer. Each type/variant/version can have its own specific management interface and semantics. New types/variants/versions are introduced all the time to handle specific situations. For example, special cells are developed to be used in trains, offices and rural areas.

---

20. p03s06.txt

Design a program that allows a network operator to plan changes in every parameter of every cell in the network. These planned changes should be verified for consistency and correctness and then applied to the network in the least disturbing way.

We would like you to pay most of your attention to the fact that there are so many different cells in the network that need to managed in a common way, but are still of very different types. The problem is that the difference cannot be abstracted away because the operator also wants to use the unique features (they want to have their cake and eat it too). Do not focus on the specific algorithms for frequency allocation and consistency checks. The program should be easily extendable to handle the algorithms and rules of the network operator so they cannot be hard-coded anyway.

Design an infrastructure where the network operator can customize critical aspects, instead of hard-coding knowledge. The program must be very flexible for changes because these networks change at a rapid pace.

---

21. p03s07.txt

Adding new cells and new cell types should be relatively easy
The operator must be able to plan to change every parameter for every cell.
It must be possible to upgrade a cell to a cell of another type and revert back. All parameters with the same name should be converted to the new type without loss of information. Defaults should be assigned to new parameters. It can of course be assumed that the program that does the conversion knows the new and old cell types.
It must be possible to develop "small" software modules that can work upon the information without getting intricately tied to all the possible cell types. Functions like: convert cell to new version.
When a change in the network fails, the plan should be rolled back to bring the network back to its former state

The program should have no restrictions on the number of cell types, or number of cells. It should be possible to minimize the number of commands sent to the cells. The reason is that certain cells have a very slow connection (seconds per command).

---

22. p03s08.txt

Each OMC has a set of BSC that it manages.
Each BSC controls one or more cells.
Each cell is on a site (and a site can have many cells on it).
Each cell has a set of neighbors, which are other cells.
Some cells are managed by the same OMC, others are managed by other OMCs or other network operators.

---

23. p03s09.txt

A cell in sector 301.402 is split in three. The original cell was a Sienokola and used an antenna that created a cell where the site was in the middle. Three directional antennas have been added plus the necessary base stations.

The operator starts the application and first selects the cell that must be removed. She indicates that the cell should be removed. She then calls up a list of neighboring cells and checks the frequencies on these cells to decide which frequencies can be used for the new cells. She then modifies the frequencies of one of the neighboring cells and creates the three new cells. The planning department had run predictions for the coverage of the new cells and had given her values for the key parameters. All the other parameters were left at default values. The next step is to see if the planned changes are consistent. This is actually the case and she orders the plan to be executed at 3 am. The next morning she finds in her mailbox a log that indicates that the changes have been successfully applied to the network.

---

24. p03s10.txt

A new supplier has entered the market that can supply base stations for a significantly lower price than the ones currently used. The network operator decides to buy 10 base stations and installs them on new sites.

The new base stations have a different management interface than the older ones. The supplier offers a special development kit that allows the base stations to be managed over a standard Simple Network Management Protocol (SNMP) interface. The development center of the network operator creates new drivers for this type and adds the type to the application. After this change, parameters can be set that are specific for the new base station.

---

25. p03s12.txt

Reports have come in that the cell in sector 231.912 is not functioning well. Many calls are dropped in a certain part of the sector. Inspection of the location reveals that a high rise was built that blocks the passage of the radio signals. Another cell 10 km to the north could reach the location but the parameter settings are such that the cell is not allowed to cover that area.

The application is started and the cell is called up. The parameters that define the area in which handovers can take place are modified. The change should take effect immediately because there are people on the location that can perform measurements. After several attempts the optimal settings are found.

---

26. p03s13.txt

The operator who orders this program has the following systems:

Sienokola.
This supplier offers cells for the GSM and DCS systems. The operator has 2000 cells of this type. Two variants are in use, the v15 type for rural areas (range 32 km) and the v16 that supports city areas (range 2 km). During the change of revision of the cells, the management software must support the new revision and the previous. The Sienokola cells can be accessed through a C-library from the Sienokola development kit.

Philison.
The operator has about 800 of these cells. This supplier offers a command line interface to the cell to set its properties. Each command starts with a name and then the parameters of the commands. Unfortunately, commands are not always "logically" constructed. A command has a maximum length of 80 characters. Commands look very similar to the AT command set for modems, for example:

(TABLE)

---

27. p03s14.txt

No two cells shall have the same name
No two neighbour cells shall use the same frequency
Cells in the same BSC should have the same MCC, LAC and MNC
Cells not in this OMC should not have the same LAC

---

28. p03s15.txt

When filter = standard, neighbour relations algorithm can only be 0 or 2
Each neighbour mentioned in a neighbour relation "rels" should have its frequencies listed in the neigbour frequencies "nfreq"
Paging channels + broadcast channels <= 12

---

29. p03s16.txt

Paging channels + broadcast channels <= 8
When bspower > 32, hopping cannot be used

---

30. p04s01.txt

Financial markets are an example of an environment where sharing and maintaining large sets of complex information give the holder competitive advantage. Often queries to such databases are imprecise, producing large result sets that are visually scanned. Also, changing market conditions and positions within them demand that results be kept current while they are displayed.

In one version of a "fat client" 128-megabyte workstations cache nearly all business objects in local memory. Logic in the client maintains the coherency of cached persistent data and merges it with continuous streams of market information broadcast through a private network.

One possible interpretation of a "thin" alternative architecture would be to move these caches and coherency logic to a small number of large middle-level servers. Modest client computers could "browse" these servers using software and protocols suggested by the world-wide web.

Appendix II

We developed the program "DetailedAnalysis.class", which trains both the sentence classifier and the discourse classifier, and then tests the final results of the discourse-level classification against the annotations of our human annotators. The log of the execution of the program is as follows (here, only the 30 passages, presented in Appendix I, are used for testing):

```
Running DetailedAnalysis.class ...
[Current Time Stamp] Sun Aug 05 07:22:04 EDT 2007

Reading Sentence-level Training Data File: sentenceLevel.arff ...done! [0.078 sec.]
Training The C4.5 Decision Tree-based Classifier ...done! [0.11 sec.]

Current C4.5 Decision Tree that classifies sentences based on ambiguity at the surface-
level is as follows:
digraph J48Tree {
N0 [label="bad_RB" ]
N0->N1 [label="<= 0"]
N1 [label="fragment" ]
N1->N2 [label="= TRUE"]
N2 [label="Ambiguous (21.89/5.0)" shape=box style=filled ]
N1->N3 [label="= FALSE"]
N3 [label="vb_in_p" ]
N3->N4 [label="<= 0"]
N4 [label="bad_DT" ]
N4->N5 [label="<= 0"]
N5 [label="Unambiguous (118.61/21.61)" shape=box style=filled ]
N4->N6 [label="> 0"]
N6 [label="adjectives" ]
N6->N7 [label="<= 1"]
N7 [label="Unambiguous (8.28/2.28)" shape=box style=filled ]
N6->N8 [label="> 1"]
N8 [label="Ambiguous (5.93)" shape=box style=filled ]
N3->N9 [label="> 0"]
N9 [label="Ambiguous (9.58/2.0)" shape=box style=filled ]
N0->N10 [label="> 0"]
N10 [label="parentheses" ]
N10->N11 [label="<= 0"]
N11 [label="bad_JJ" ]
N11->N12 [label="<= 0"]
N12 [label="Unambiguous (4.82/0.82)" shape=box style=filled ]
N11->N13 [label="> 0"]
N13 [label="Ambiguous (13.65/3.0)" shape=box style=filled ]
N10->N14 [label="> 0"]
N14 [label="Ambiguous (53.22/1.0)" shape=box style=filled ]
}

Stanford Lexicalized Parser v1.5.1 is now loaded into memory.
```

```
Loading Keyword Lists ...done! [0.016 sec.]

Reading Discourse-level Training Data File: discourseLevel.arff ...done! [0.031 sec.]
Training The C4.5 Decision Tree-based Classifier ...done! [0.07 sec.]

Current C4.5 Decision Tree that classifies a discourse based on ambiguity at the surface-
level is as follows:
digraph J48Tree {
N0 [label="ambiguity_density" ]
N0->N1 [label="<= 0"]
N1 [label="Unambiguous (52.64/0.64)" shape=box style=filled ]
N0->N2 [label="> 0"]
N2 [label="Ambiguous (112.36/19.0)" shape=box style=filled ]
}



   ==================================
         ANALYZING PERFORMANCE
   ==================================



Reading human annotations from: bin/data/annotations.csv

--------------------------------------------------
Processing file #1: p01s01.txt...done! [4.36 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- As the main incoming channel for cases is frequently the telephone, the system must
have sub-second response times and be intuitive to use.
*** The sentence contains:
         Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #2: p01s02.txt...done! [18.578 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- Prioritise the cases based on specific rules (including Artificial Intelligence engines
that deal with complex rules).
*** The sentence contains:
         Adjective(s), Word(s) inside Parentheses, Verb(s) inside Parenntheses,
Parentheses, Ambiguous Adjectives, .

- The design problem for DesignFest® is to create an Object Oriented framework for use in
developing Case Management applications quickly, and customisable to the particular work
practices of the company purchasing the framework.
*** The sentence contains:
         Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #3: p01s03.txt...done! [2.906 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:
```

```
- There is no requirement for this to be configurable at run time (but it wouldn hurt if
that was an option).
*** The sentence contains:
        Adjective(s), Word(s) inside Parentheses, Verb(s) inside Parenntheses,
Parentheses, Ambiguous Adjectives, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #4: p01s04.txt...done! [18.469 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- The manager then determines the next appropriate step (close, assign to another case
worker for the next processing step, put aside until a bring-forward is activated, etc.).
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Word(s) inside Parentheses,
Verb(s) inside Parenntheses, Parentheses, Ambiguous Adjectives, .

- A case worker selects cases from a pool to perform a specific function, and then moves
that case to the next appropriate step (close, put in another pool, put aside until a
bring-forward is activated, etc.).
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Word(s) inside Parentheses,
Verb(s) inside Parenntheses, Parentheses, Ambiguous Adjectives, .

- Finally, workflow support also means providing the ability to Approve or Reject a
particular request regarding a case.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .

- A case may involve a single or multiple requests requiring approval, and security may
be needed to restrict who can make these decisions, depending on the organisation
building the application using this framework.
*** The sentence contains:
        Adjective(s), Verb(s) in Passive form, Ambiguous Determinants, Ambiguous Modals,
Ambiguous Adjectives, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #5: p01s05.txt...done! [7.234 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- Changing to a particular status may cause an event to be created automatically, such as
creating a bring-forward entry due at some future date.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Ambiguous Determinants,
Ambiguous Modals, Ambiguous Adjectives, Ambiguous Adverbs, .

- Changing to a particular status may also automatically close a bring-forward (whether
it is due or not).
*** The sentence contains:
        Adjective(s), Adverb(s), Word(s) inside Parentheses, Verb(s) inside Parenntheses,
Parentheses, Ambiguous Modals, Ambiguous Adjectives, Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------
```

```
--------------------------------------------------
Processing file #6: p01s06.txt...done! [0.766 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #7: p01s07.txt...done! [5.5 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- If case worker are geographically dispersed, then you may assume that the cases are
managed by geographic regions.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Ambiguous Modals, Ambiguous
Adjectives, Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #8: p02s01.txt...done! [11.547 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #9: p02s02.txt...done! [5.515 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- The production is planned at factory level and is represented by lots assigned to the
FMS (a lot is an administrative entity indicating a group of identical pieces to be
manufactured together.) Several lots of different types of pieces can be in production at
the same time in a given FMS.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Too Many Words, Word(s) inside
Parentheses, Verb(s) inside Parenntheses, Parentheses, Ambiguous Adjectives, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #10: p02s03.txt...done! [12.297 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- Machines are assigned to a waiting piece (if any) as soon as they become idle. A
machine is idle when it is not performing an operation on a piece and the last piece
operated on has been evacuated.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Word(s) inside Parentheses,
Parentheses, Ambiguous Determinants, Ambiguous Adjectives, .

- For our purposes (as is the case in a small FMS), the time needed to serve a transport
request is small relative to the time needed for a machine operation.
*** The sentence contains:
```

```
        Adjective(s), Word(s) inside Parentheses, Verb(s) inside Parenntheses,
Parentheses, Ambiguous Adjectives, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #11: p02s04.txt...done! [20.203 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- Two turning machines, M1 and M2, and a drilling machine, M3, each of which can work on
a single piece at a time.
*** The sentence contains:
        Adjective(s), Ambiguous Adjectives, The sentence is detected as fragment.

- Two types of pieces, TP1 and TP2.
*** The sentence contains:
        The sentence is detected as fragment.

- The store for temporary in-process pieces: Store_tmp.
*** The sentence contains:
        Adjective(s), Ambiguous Adjectives, The sentence is detected as fragment.

- The transportation system with three carts, each of which can carry a single piece and
is equipped with a robot arm that loads or unloads the cart on command from the FMS
computer.
*** The sentence contains:
        Adjective(s), Verb(s) in Passive form, Ambiguous Adjectives, The sentence is
detected as fragment.

- The controller/dispatcher knows at all times the state of all of the machines, the
places in the stores and the carts (free or occupied, and if occupied, by which piece) as
well as the position and state of all the individual pieces.
*** The sentence contains:
        Adjective(s), Adverb(s), Too Many Words, Word(s) inside Parentheses, Verb(s)
inside Parenntheses, Parentheses, Ambiguous Determinants, Ambiguous Adjectives, Ambiguous
Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #12: p02s05.txt...done! [4.063 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #13: p02s06.txt...done! [8.281 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: MISCLASSIFICATION ->  System: Unambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #14: p02s07.txt...done! [1.109 sec.]
The Passage is classified as 'Unambiguous'.
```

```
   --- Result: MISCLASSIFICATION ->  System: Unambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #15: p03s01.txt...done! [4.563 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- It used to be possible to configure these networks by hand, but the tremendous growth
has made this infeasible.
*** The sentence contains:
        Adjective(s), Ambiguous Determinants, Ambiguous Adjectives, .

- There are just too many cells to be handled by man.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Ambiguous Adjectives,
Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #16: p03s02.txt...done! [6.422 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- When a network is initially built, the area that is covered by each cell is very large.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Ambiguous Adjectives,
Ambiguous Adverbs, .

- This is almost a black art and is usually handled by special planning programs.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Ambiguous Adjectives,
Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #17: p03s03.txt...done! [2.843 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #18: p03s04.txt...done! [2.625 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- The sites are the most visible part of a cellular network because of the very obvious
antenna masts that are visible all over the country.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Determinants, Ambiguous Adjectives, Ambiguous
Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------
```

```
--------------------------------------------------
Processing file #19: p03s05.txt...done! [11.532 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- Planning, subscriptions, procurement, installation, handling of alarms, measurements,
configuration changes and many more.
*** The sentence contains:
        Adjective(s), Ambiguous Adjectives, The sentence is detected as fragment.

- The networks have become so large that they are normally split sections.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Ambiguous Adjectives,
Ambiguous Adverbs, .

- This is complicated because each cell has literally hundreds of parameters. A short
summary:
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .

- Relationship to neighbouring cells.
*** The sentence contains:
        The sentence is detected as fragment.

- Access rights from mobile phones.
*** The sentence contains:
        Adjective(s), Ambiguous Adjectives, The sentence is detected as fragment.

- Several of these parameters are dependent on, and constrained by, the settings of other
parameters in other cells.
*** The sentence contains:
        Adjective(s), Ambiguous Determinants, Ambiguous Adjectives, .

- For example, neighboring cells should always use different frequencies.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Modals, Ambiguous Adjectives, Ambiguous
Adverbs, .

- Changing parameters in the network must be done in a short time on all related cells to
keep the disturbance to the network as low as possible.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Ambiguous Determinants,
Ambiguous Adjectives, .

- Therefore, the changes must be planned and executed in a short time, usually during the
middle of the night.
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Ambiguous Adjectives,
Ambiguous Adverbs, .

- A very complicating factor in the management of the cells is that in one network, one
can find cells from different manufacturers and different versions and variants from the
same manufacturer.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #20: p03s06.txt...done! [6.312 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:
```

```
- We would like you to pay most of your attention to the fact that there are so many
different cells in the network that need to managed in a common way, but are still of
very different types.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .

- The problem is that the difference cannot be abstracted away because the operator also
wants to use the unique features (they want to have their cake and eat it too).
*** The sentence contains:
        Adjective(s), Adverb(s), Verb(s) in Passive form, Word(s) inside Parentheses,
Verb(s) inside Parenntheses, Parentheses, Ambiguous Adjectives, Ambiguous Adverbs, .

- The program must be very flexible for changes because these networks change at a rapid
pace.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Determinants, Ambiguous Adjectives, Ambiguous
Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #21: p03s07.txt...done! [4.266 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- Adding new cells and new cell types should be relatively easy.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Modals, Ambiguous Adjectives, Ambiguous
Adverbs, .

- All parameters with the same name should be converted to the new type without loss of
information.
*** The sentence contains:
        Adjective(s), Verb(s) in Passive form, Ambiguous Determinants, Ambiguous
Modals, .

- It must be possible to develop "small" software modules that can work upon the
information without getting intricately tied to all the possible cell types.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .

- The reason is that certain cells have a very slow connection (seconds per command).
*** The sentence contains:
        Adjective(s), Adverb(s), Word(s) inside Parentheses, Parentheses, Ambiguous
Adjectives, Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #22: p03s08.txt...done! [1.234 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- Each cell is on a site (and a site can have many cells on it).
*** The sentence contains:
        Adjective(s), Word(s) inside Parentheses, Verb(s) inside Parenntheses,
Parentheses, Ambiguous Adjectives, .

- Some cells are managed by the same OMC, others are managed by other OMCs or other
network operators.
*** The sentence contains:
        Adjective(s), Verb(s) in Passive form, Ambiguous Determinants, .
```

```
   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #23: p03s09.txt...done! [5.516 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #24: p03s10.txt...done! [3.375 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #25: p03s12.txt...done! [3.079 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #26: p03s13.txt...done! [3.89 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- Two variants are in use, the v15 type for rural areas (range 32 km) and the v16 that
supports city areas (range 2 km).
*** The sentence contains:
        Adjective(s), Word(s) inside Parentheses, Verb(s) inside Parenntheses,
Parentheses, .

- Unfortunately, commands are not always "logically" constructed.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .

- Commands look very similar to the AT command set for modems, for example:
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Adjectives, Ambiguous Adverbs, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #27: p03s14.txt...done! [0.688 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #28: p03s15.txt...done! [1.218 sec.]
```

```
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #29: p03s16.txt...done! [0.282 sec.]
The Passage is classified as 'Unambiguous'.


   --- Result: CORRECT ->  System: Unambiguous  |  Annotator: Unambiguous
--------------------------------------------------


--------------------------------------------------
Processing file #30: p04s01.txt...done! [4.953 sec.]
The Passage is classified as 'Ambiguous'.
The following Ambiguous Sentence(s) are detected:

- In one version of a "fat client" 128-megabyte workstations cache nearly all business
objects in local memory.
*** The sentence contains:
        Adjective(s), Adverb(s), Ambiguous Determinants, Ambiguous Adjectives, Ambiguous
Adverbs, .

- One possible interpretation of a "thin" alternative architecture would be to move these
caches and coherency logic to a small number of large middle-level servers.
*** The sentence contains:
        Adjective(s), Ambiguous Determinants, Ambiguous Adjectives, .

- Modest client computers could "browse" these servers using software and protocols
suggested by the world-wide web.
*** The sentence contains:
        Adjective(s), Ambiguous Determinants, Ambiguous Modals, Ambiguous Adjectives, .


   --- Result: CORRECT ->  System: Ambiguous  |  Annotator: Ambiguous
--------------------------------------------------




End of analysis.
----------------


DetailedAnalysis.class: Finished
[Current Time Stamp] Sun Aug 05 07:45:31 EDT 2007
```

Appendix III

The following are snapshots of the user interface of our prototype, called *ReqSAC* (Requirements Specification Ambiguity Checker):

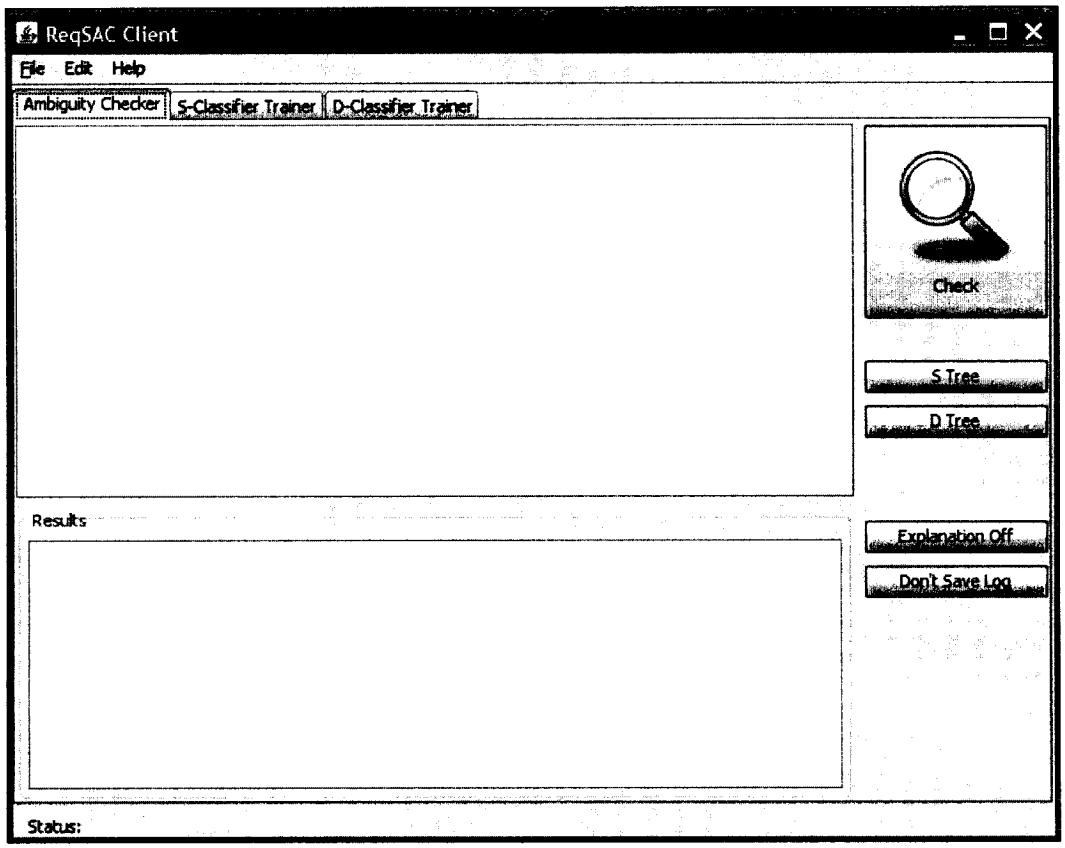

Figure 31: A snapshot of the main window of ReqSAC

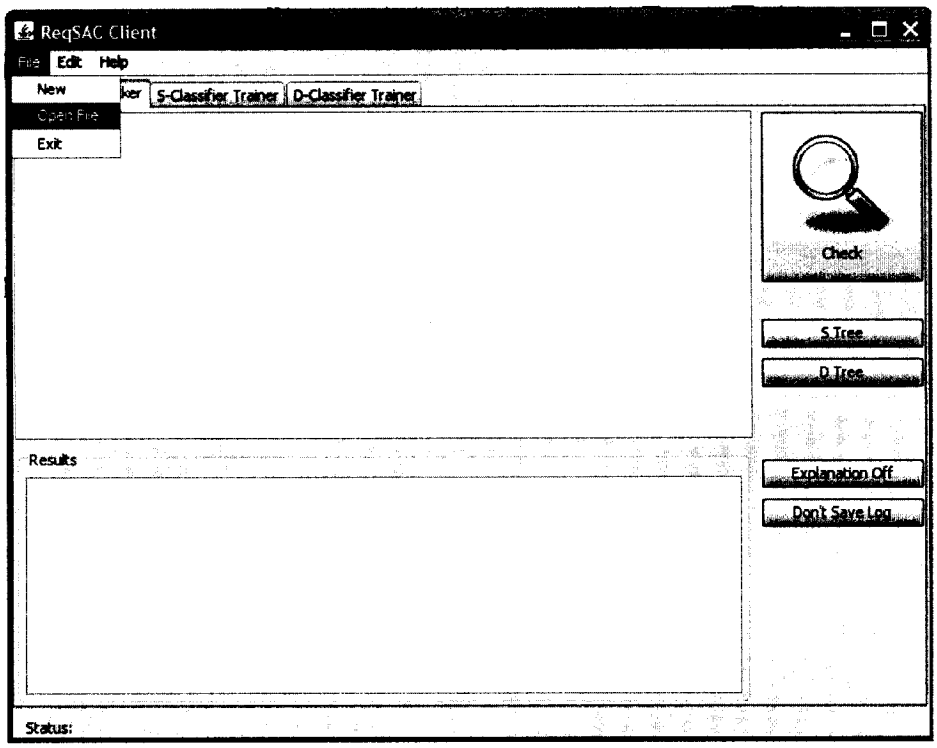

Figure 32: A snapshot of ReqSAC showing the *File* menu

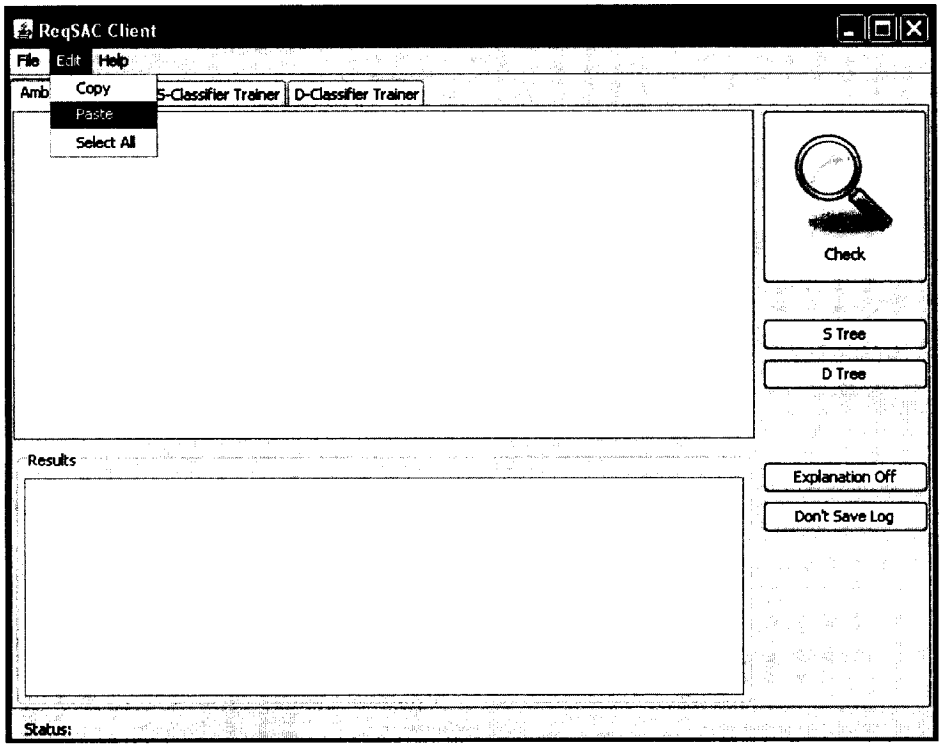

Figure 33: A snapshot of ReqSAC showing the *Edit* menu

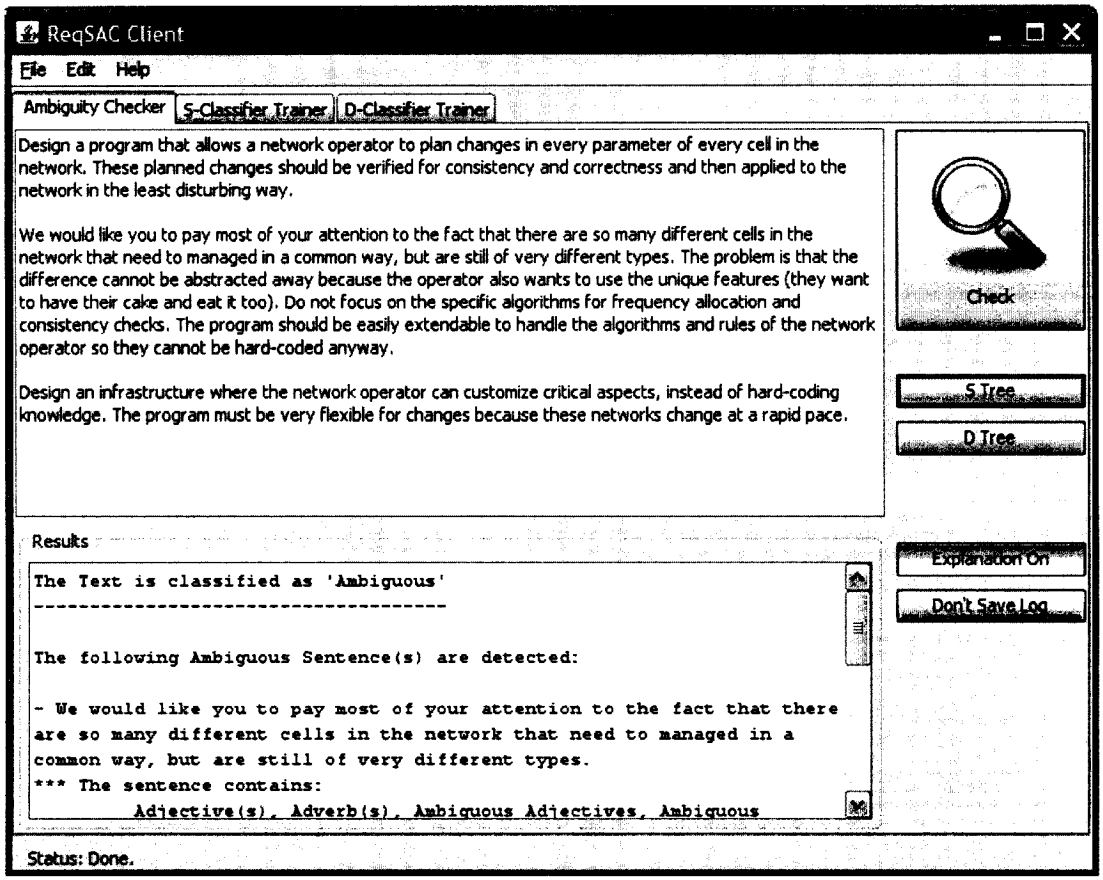

Figure 34: A snapshot of ReqSAC checking the ambiguity of a text with "*Explanation On*"

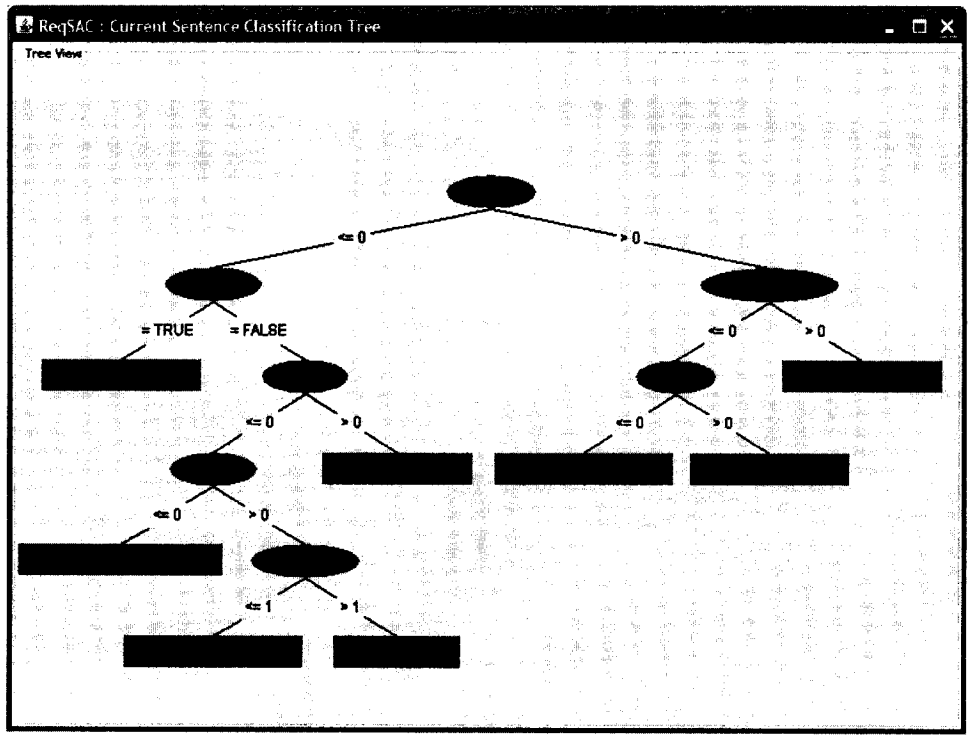

Figure 35: A snapshot of the Sentence Classification Tree viewer window of ReqSAC

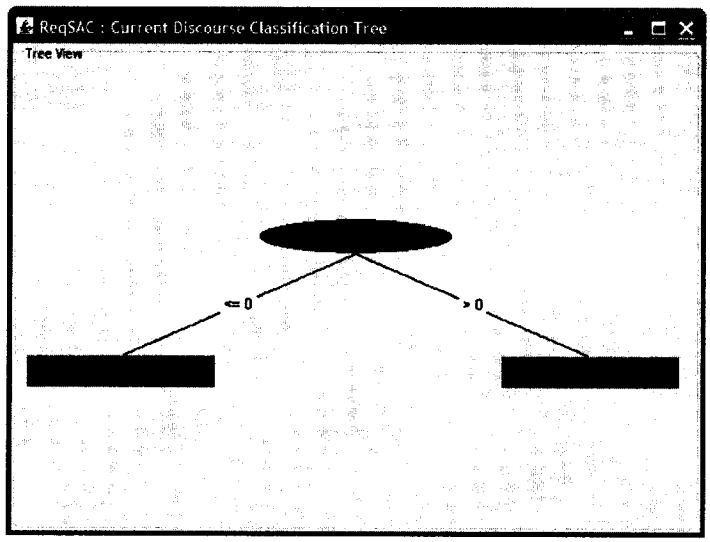

Figure 36: A snapshot of the Discourse Classification Tree viewer window in ReqSAC