Automatic Detection of Nocuous Coordination Ambiguities in Natural Language Requirements

Hui Yang[1] Alistair Willis[1] Anne De Roeck[1] Bashar Nuseibeh[1,2]

[1]Department of Computing
The Open University
Milton Keynes, MK7 6AA, UK
{h.yang, a.g.willis, a.deroeck, b.nuseibeh}@open.ac.uk

[2]Lero
University of Limerick
Limerick, Ireland
Basher.Nuseibeh@iero.ie

ABSTRACT

Natural language is prevalent in requirements documents. However, ambiguity is an intrinsic phenomenon of natural language, and is therefore present in all such documents. Ambiguity occurs when a sentence can be interpreted differently by different readers. In this paper, we describe an automated approach for characterizing and detecting so-called *nocuous ambiguities*, which carry a high risk of misunderstanding among different readers. Given a natural language requirements document, sentences that contain specific types of ambiguity are first extracted automatically from the text. A machine learning algorithm is then used to determine whether an ambiguous sentence is nocuous or innocuous, based on a set of heuristics that draw on human judgments, which we collected as training data. We implemented a prototype tool for Nocuous Ambiguity Identification (NAI), in order to illustrate and evaluate our approach. The tool focuses on *coordination ambiguity*. We report on the results of a set of experiments to assess the performance and usefulness of the approach.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specification**]: Elicitation Methods, Language, Methodologies, Tools

General Terms

Management, Measurement, Performance, Experimentation

Keywords

Natural language requirements, nocuous ambiguity, coordination ambiguity, machine learning, human judgments

1. INTRODUCTION

Natural language (NL) is still prevalent in the vast majority of requirements documents [3]. One important reason for this is that NL can help various stakeholders articulate and communicate requirements during the entire life cycle of the software development. However, NL requirements also suffer from typical NL problems such as ambiguity. Ambiguity occurs when a single linguistic expression can be interpreted differently by different readers. Ambiguous expressions in requirements can be potentially dangerous when they result in poor requirements quality [4]. Our research is motivated by the need to reduce the costs of misunderstandings that can occur during requirements engineering, when these misunderstandings are due to ambiguities in the NL requirements. Our practical goal is to provide a tool to assist writers of requirements documents by alerting them to potentially harmful ambiguities, called *nocuous* ambiguities [6]. Unlike innocuous ambiguities, which tend to be interpreted in the same way by all readers, nocuous ambiguities give rise to different interpretations by different readers, thus contributing to misunderstandings between stakeholders. Such a tool needs to highlight those linguistic expressions in requirements that are recognized as nocuous ambiguities, and allow the writers to return to elicitation, or rephrase for the purpose of improving requirements quality, and facilitating effective communication of these requirements among different stakeholders.

In earlier work [25], we proposed a general methodology for automatic identification of nocuous ambiguity, which we use to guide our research on two types of ambiguity, coordination ambiguity [6, 22] and anaphora ambiguity [24]. In contrast to other work, which is intended to resolve ambiguity [5, 18], our research concerns identification of those ambiguities that are likely to lead to misunderstandings between different readers, while discounting those which tend to be interpreted in the same way by different readers despite their surface features. As such, we consider ambiguity as a property of the relationship between a text and a group of interpreters, rather than a property of a text or expression *per se*. We also add the categorization of *nocuous* and *innocuous* ambiguity depending on the likely distribution of interpretations held by a group of readers of that text. We have observed that not all cases of the ambiguity are actually dangerous: in fact, most remain unnoticed and are resolved to the same interpretation by all stakeholders. Only *nocuous* ambiguity cases that have a high risk of misunderstanding between different readers are truly disruptive and deserving of further attention.

Our previous work [6, 22] focused on coordination ambiguity, a particularly common kind of structural ambiguity, highly prevalent in requirements documents. We investigated a methodology that used a number of heuristics based on corpus-based statistical information together with human judgments to predict whether a

coordination ambiguity may be misunderstood for a given ambiguity threshold. However, our first system was semi-automatic: it relied on manual selection of coordination ambiguity instances from a number of requirements documents. In this paper, we extend our previous work on coordination ambiguity by introducing new functional process modules, such as the extraction of ambiguous coordination instances and the recognition of coordination constituents contained in coordination constructions. These are necessary for automatic identification of nocuous coordination ambiguity. Moreover, we implemented a prototype tool, called Nocuous Ambiguity Identification (NAI), to illustrate and evaluate our approach. Given an NL requirements document, our tool can generate a workflow that integrates with several required functional modules to automatically detect nocuous ambiguity.

The work in this paper differs from our previous work [6, 22] in the following four ways: first, we bring in information extraction techniques to tackle some problems in automatic detection of coordination ambiguity, which include pattern-based matching to detect ambiguous instances contained in the sentences, and named entity recognition (NER) techniques for the extraction of coordination constituents, such as coordinating conjuncts and the attached modifiers. Second, we introduce two more heuristics, collocation frequency in the local document and semantic similarity. These enrich previous heuristics and explore further aspects of coordination ambiguity that may lead a reader to prefer a particular reading. Third, we employ the LogitBoost algorithm for the building of a so-called "nocuity classifier". We show that this machine learning algorithm performs better than the Logistic Regression algorithm that was used in our previous work [22]. Fourth, we implement an automated tool to detect and highlight on screen, nocuous ambiguity in text. We are not aware of any comparable automated tools.

The rest of the paper is structured as follows. In Section 2, we introduce the ambiguity problem in requirements documents. Section 3 provides the detailed description of individual functional modules in the framework of the NAI tool used for coordination ambiguity. The building of a coordination ambiguity dataset and the construction of the nocuity classifier are described in Section 4. Experimental setup and results are reported in Section 5. Section 6 discusses related work, and conclusions and future work are presented in Section 7.

2. The Ambiguity Problem

Ambiguity is a phenomenon inherent in natural language. It occurs everywhere in natural language requirements. For example, in a total of 11 requirements engineering documents we collected for the study, there were 3404 sentences containing coordination ambiguity instances, which make up about 12.68% of all the sentences (26829 in total) in this dataset.

Computationally, structural ambiguity can be recognised when a parser assigns more than one possible parse to a sentence. Coordination ambiguity is one of the particularly common kinds of structural ambiguity. A coordination structure connects two words, phrases, or clauses together via a coordination conjunction (e.g., '*and*', '*or*', etc). Consider the following real examples[1]:

E1. *They support a typing system for architectural components and connectors.*

E2. *It is described the size of vector-based input and output.*

In the example (E1), the coordination construction '*architectural components and connectors*' can be bracketed as '*[architectural [components and connectors]]*' that is interpreted as '*architectural components*' and '*architectural connectors*', or as '*[architectural components] and [connectors]*', in which case it is only *component* which is modified by *architectural*. However, in the example (E2), although the coordination construction '*vector-based input and output*' can also be interpreted as '*[vector-based [input and output]]*' or '*[vector-based input] and [output]*', it seems that the former reading is most likely to be preferred, perhaps because of semantic similarity between input and output.

When presented to human judges, for example (E1), 7 out of 17 judges committed to the reading '*[architectural [components and connectors]]*' whereas 7 of the remaining judges chose the reading '*[architectural components] and [connectors]*' (information about human judgment collection is given in the section 4.1). We treat (E1) as an example of *nocuous ambiguity* because the risk of diverging interpretation is high. In contrast, for the example (E2), the majority of judges (16 out of 17) agreed with the reading '*[vector-based [input and output]]*. So we say (E2) exhibits *innocuous ambiguity*, because it is interpreted in the same way by different readers, and has a low risk of being misunderstood.

Table 1. Construction patterns used in coordination ambiguity (n_1 and n_2, v_1 and v_2 are coordinated compounds (i.e. noun or verb compound); the underline part-of-speech tag is the headword of the attached modifier; c is a coordination (i.e. '*and*' or '*or*'); p is the preposition)

Type	Pattern	Example
noun	adj n_1 c n_2	manual *input* and *output*
	vbn n_1 c n_2	associated *doors* and *windows*
	nn n_1 c n_2	project *manager* and *designer*
	[dtladj] nn p n_1 c n_2	the set of *plans* and *tables*
	n_1 c n_2 nn	*software* and *hardware* product
	n_1 c n_2 p [dtladj] nn	*book* and *paper* on the table
verb	adv v_1 c v_2	be manually *rejected* and *flagged*
	[dtladj] nn p v_1 c v_2	some functions for *receiving* and *transmitting*
	v_1 c v_2 nn	*generate* and *print* reports
	v_1 c v_2 adv	be *inspected* and *recorded* automatically
	v_1 c v_2 p [dtladj] nn	be *implemented* and *executed* on the platform

[1] Our examples are modified extractions from requirements documents available on the web site, http://research.it.uts.edu.au.

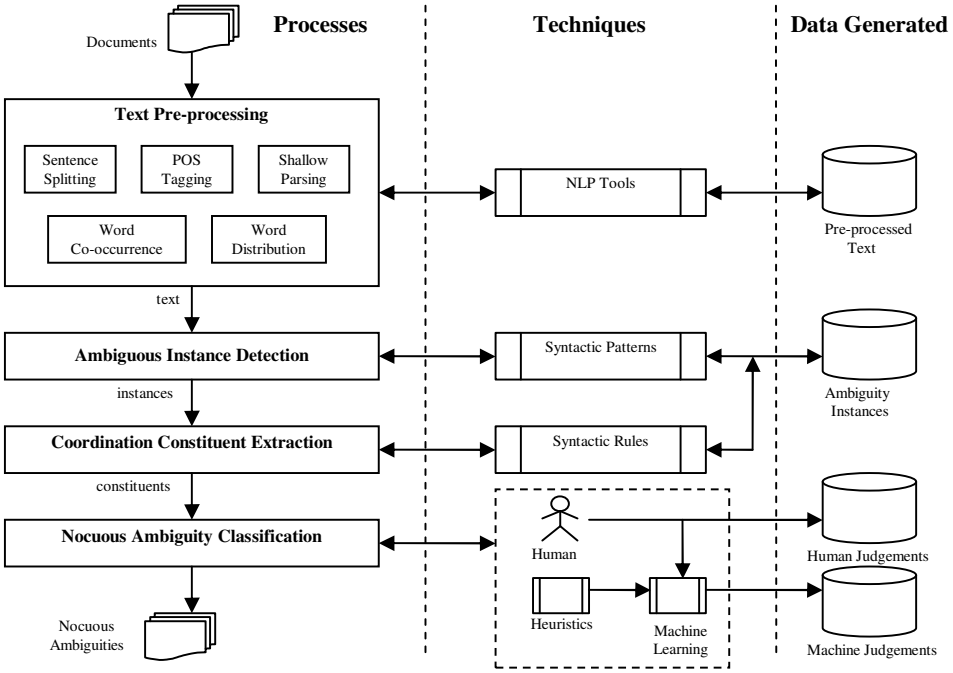

Figure 1. The framework for the NAI tool used in coordination ambiguity

Our automated approach focuses on two main types of coordination ambiguity: noun compound coordination and verb compound coordination, respectively. The construction patterns for these two types of coordination ambiguity are depicted in Table 1. The table shows that each ambiguity construction pattern generally consists of three basic *coordination constituents*, two coordinating conjuncts, near conjunct (NC) and far conjunct (FC), and the attached modifier (F). The conjunct is allowed to be a noun or verb compound, and the modifier to be an adjective, adverb, or noun.

For example, the coordination construction '*functional model and description*' is the exemplification of the ambiguity pattern '*adj* n_1 *c* n_2', where '*model*' and '*description*' are NC and FC, and the adjective '*functional*' is the M, individually. Syntactically, this pattern can be interpreted by two distinct bracketing: (1) '*[adj [*n_1 *c* n_2*]]*' - '*[functional [model and description]]*' and (2) '*[adj* n_1*]c [*n_2*]*' - '*[functional model] and [description]*'. We say that (1) displays *high attachment* of the modifier, where M applies to both FC and NC, and (2) displays *low attachment*, where M applies only to NC.

3. A Nocuous Ambiguity Identification Tool

We have implemented a Nocuous Ambiguity Identification (NAI) tool to automatically detect nocuous ambiguities in text. NAI identifies potentially ambiguous patterns in a textual input, and a nocuity classifier then classifies the instances into nocuous and innocuous cases.

The conceptual architecture of the NAI tool is shown in Figure 1. It consists of four main functional process modules: Text Pre-processing Module, Ambiguity Instance Detection Module, Coordination Constituent Extraction Module, and Nocuous Ambiguity Classification Module, respectively. We describe the behavior of each of the modules in turn, and give details of how the system is trained in Section 4.

3.1 Text Pre-processing

Given a text document, our tool first executes several text pre-processing steps, including sentence splitting, part-of-speech (POS) tagging, and phrase-based shallow parsing. At first, the text is split into a collection of sentence using a sentence boundary detector[2]. Then, for each sentence, the Stanford parser[3] is used to obtain word lemma and POS tags (e.g., noun, verb, adjective, adverb, etc.) of individual words and associated phrase information. Furthermore, some statistical information (e.g., word co-occurrence and word distribution) is calculated based on the position of words in the sentence, and saved into the back-end MySQL database.

3.2 Ambiguous Instance Detection

This process consists of two main steps: Step I is to detect the sentences that contain coordination constructions; Step II is to extract relevant ambiguity instances from the detected sentences.

Given a sentence, Step I is to search the POS-tag sequence of the sentence and determine whether the sentence contains one of the coordination construction patterns described in Table 1. A searching window is moved from left to right to examine the POS-tag sequence of the sentence. If an ambiguity construction pattern falls into the window, then the corresponding substring is extracted from the sentence as an ambiguous coordination instance.

[2] http://text0.mib.man.ac.uk:8080/scottpiao/sent_detector

[3] http://nlp.stanford.edu/software/lex-parser.shtml

If the sentence does contain one or more of the patterns, then step II extracts relevant coordination constructions from the sentences. This step is fundamental to the understanding of a sentence, especially for a sentence which is longer and far more complex [1]. For example,

E3. *Use daylight to achieve the desired light setting of room and hallway section whenever possible.*

In the sentence (E3), the substring '*the desired light setting of room and hallway section*', in practice, can be split into two distinct ambiguity instances, '*the desired light settings of room and hallway*' and '*room and hallway section*'. To simplify the ambiguity analysis, it is necessary to treat these two ambiguity instances separately, and determine whether either is nocuous. Therefore, our ambiguity analysis is based on a smaller unit level - the instance level other than on the sentence level with respect to long and complicated sentences.

3.3 Coordination Constituent Extraction

The recognition of coordination constituents is, in fact, the task of named entity recognition (NER). NER plays an important role in analyzing the syntactic and semantic relations among coordination constituents discussed later. In order to extract appropriate coordination constituents from an ambiguous construction pattern, a set of syntactic rules are created based on the position of individual constituents in the pattern and the property of their corresponding POS tags. For each ambiguity instance, two coordinating conjuncts, near conjunct (NC) and far conjunct (FC), and the attached modifier (M) are separately extracted from the instance by relevant syntactic rule, and saved into the database together with the corresponding instance id.

3.4 Heuristics to Predict Nocuity

Having identified the sentences which contain instances of coordination ambiguity, and recognized the constituents which give rise to the ambiguity, the final stage is to identify whether the ambiguity is nocuous or not. To identify cases of nocuous ambiguity, we implemented a number of heuristics that identify the properties of an ambiguity instance, which in turn may lead an interpreter to favor high attachment or low attachment interpretations. Our tool then classifies the input as nocuous or innocuous using the values of the set of heuristics input to a trained classifier using the LogitBoost algorithm. The remainder of this subsection deals with the particular heuristics; details of training and using the classifier are given in section 4.

We explored a number of heuristics to apply to ambiguity instances. Each of the individual heuristics attempts to identify aspects of the ambiguity instance that may lead an interpreter to favor high attachment or low attachment interpretation. Heuristics run over the instance and their scores are saved as features in a feature vector that will be used later by a machine learning algorithm to classify coordination ambiguity as nocuous or innocuous.

One of the major approaches we use here is a corpus-based approach. Some of the heuristics (e.g., coordination matching, distribution similarity, and collocation frequency) are based on statistical information (e.g., word distribution and collocation) that is obtained from a large English corpus – British National Corpus[4] (BNC) via the Sketch Engine[5] [14]. We here give a brief description of each of the heuristics below. Besides the four heuristics used in the previous work [6], we introduce two more heuristics, collocation frequency in local document and WordNet-based semantic similarity. We present below a brief description of each of the heuristics.

Coordination Matching. This heuristics hypothesize that if the headwords[6] of the two conjuncts are frequently coordinated in the text, then that coordination forms a single syntactic unit, and the particular instance should therefore prefer to *high attachment* interpretation. The word sketch facility of the Sketch Engine provides statistical information about lists of words that are conjoined with '*and*' or '*or*'. The higher the ranked score is, the more frequently the two conjuncts occur in the BNC corpus. Consider the example below:

E4. *Security and Privacy Requirement*

The Sketch Engine returned a highly-ranking score of 5.8 for the coordinated words '*security*' and '*privacy*'. In the human judgments we collected, 12 out of 17 judges considered that it should be interpreted as high attachment interpretation, 4 thought it ambiguous, and only 1 judge chose low attachment interpretation. Therefore, this ambiguity tends to be interpreted as a high attachment reading, i.e. '*[[Security and Privacy] Requirement]*'.

Distributional Similarity. This heuristic is based on the assumption suggested by Kilgarriff [13], which is that strong distributional similarity between the headwords of the two conjuncts indicates that the conjuncts form a syntactic unit, thus resulting in the preference of the *high attachment* interpretation. The distributional similarity of two words is a measure of how often these two words can be found in the same contexts. For example,

E5. *Function for receiving and transmitting*

The words 'receiving' and 'transmitting' have strong distributional similarity despite their opposite meanings. Of the 17 judges, 14 judged this ambiguity to be high attachment interpretation, 3 judged it to be an ambiguity, but no one considered it a low attachment reading.

Collocation Frequency. This heuristic assumes that if the modifier is collocated much more frequently with the headword of the near conjunct than it is collocated with the headword of the far conjunct; the particular instance should display *low attachment* of the modifier. The score returned by the heuristics is the ration of the collocation frequency with the near headword over the collocation frequency with the further headword. Collocation frequency had been proven very useful in the disambiguation task [18, 21]. For instance,

E6. *Project manager and designer*

'*project*' has a collocation score of 29.55 with '*manager*' in the BNC, but it has no collocation with '*designer*'. In the collected

[4] http://www.natcorp.ox.ac.uk/

[5] http://www.sketchengine.co.uk/

[6] A *headword* is the main word of a phrase, and the other words in that phrase modify it.

judgments, 8 judges favored low attachment reading while 4 people preferred high attachment reading. It seems this ambiguity is more likely to be interpreted as low a attachment reading.

Here this heuristic was employed on two different resources and obtained separate heuristics scores:

a) Local document: the *local-based* collocation frequency score is calculated based on the collocation frequency information in the local document. This score looks more useful especially when the headwords for coordination constituents are domain-specific words that are too scarce to be found in the BNC. Moreover, the contexts in the local document probably provide stronger cues for the semantic relationships between the conjuncts and the attached modifier.

b) BNC Corpus: the *corpus-based* collocation frequency score that is estimated based on the collocation frequency information on the BNC corpus.

Morphology. This heuristic is based on syntactic parallelism suggested by Okumura and Muraki [19] in disambiguating coordination. It hypothesizes that if the headwords of the two conjuncts share a similar morphology, then they form a syntactic unit, hence the instance favoring the high attachment interpretation. The inflectional morphology of a language is the analysis of the changing of words to signify their term, number, gender etc: in English it consists largely of suffixes such as –ed, -ing, and -s. The derivational morphology of English is more complex, but suffixes, such as -ation and -able, are also very common. The score returned by this heuristic is the number of common trailing characters of the headwords of the potential conjuncts.

Semantic Similarity. This heuristic presents a measure of semantic similarity between the headwords of the two conjuncts based on the taxonomic structure in WordNet[7]. Resnik [20] has pointed out that similarity of meanings of conjoined heads is an important cue to coordination ambiguity resolution. For instance,

E7. *vector-based input and output*

E8. *manual input and selection*

Clearly '*input*' and '*output*' in (E7) are more similar in semantics than '*input*' and '*selection*' in (E8). Therefore, (E7) could be more likely to be interpreted as *high attachment* of the modifier. This assumption conforms to the distribution of human judgment. In (E7), 16 out of 17 judges had a favor of high attachment reading, while in (E8), only 5 of 17 judges chose high attachment interpretation.

The similarity of meaning could be captured well by semantic similarity in the WordNet taxonomy by measuring the distance between the nodes corresponding to the headwords of the two conjuncts. If the headwords of coordinating conjuncts exhibit strong semantic similarity, then the ambiguity instances favour a high attachment interpretation. This heuristic is implemented by the NLP tool - Java WordNet Similarity Library[8].

[7] http://wordnet.princeton.edu/

[8] http://nlp.shef.ac.uk/result/software.html

4. Training and Using the Nocuity Classifier

As discussed in section 3.4, each of the heuristics individually indicates a preference for high or low attachment. However, they do not predict whether a given ambiguity instance is nocuous or not. In this section, we describe how the individual heuristics are combined to classify a particular ambiguity as nocuous or innocuous at a given ambiguity threshold.

4.1 Building a Dataset

In order to build a working classifier, we require a collection of human judgments of ambiguous sentences. This allows us to identify which sentences display nocuous ambiguity, and use this information to identify how the particular heuristics described in section 3.4 are combined to replicate the human judgments.

To train NAI to recognise instances of nocuous ambiguity, we used the dataset collected by Chantree et al. [6]. This dataset describes 138 coordination instances from the sentences of a set of requirement documents. Each of the instances contains one of the ambiguity construction patterns described in Table 1. Among the instances, noun compound conjunctions account for a significant number, with 118 instances (85.5%). In noun compound conjunctions, nearly half of the cases arose as a result of noun modifiers, while there are 36 cases with adjective and 18 with preposition modifiers.

Human Judgment Collection. The coordination instances that contain potential ambiguity were split into 4 surveys and presented to a group of computing professionals including academic staff or research students. Each instance was judged by 17 people. For each instance, the judges were asked to select one of the three options: high attachment (HA) of the modifier, low attachment (LA) of the modifier, or ambiguous (A). The latter we call 'acknowledged ambiguity' - i.e. the reader realizes an ambiguity is present in the text, but does not feel able to judge which interpretation was intended by the writer. Table 2 shows the judgment count for two sample instances. Instance (a) was judged mainly to have high attachment of the modifier, while instance (b) was judged mainly to be ambiguous.

Table 2. Judgment count for the sample instances (HA=high attachment; LA=low attachment; and A=Ambiguous)

	Judgments		
	HA	LA	A
(a) *security* and *privacy* requirements	12	1	4
(b) electrical *characteristics* and *interface*	4	4	9

4.2 Training the Classifier

A key concept in training the classifier is the *ambiguity threshold*. We require a decision point to determine whether a particular instance exhibits nocuous ambiguity or not. We use the concept of *ambiguity threshold* that represents how much agreement is required from the judges over a particular interpretation. Use of a threshold also allows us to adjust tolerance levels: some application domains (e.g., safety critical systems) may wish to use to a low threshold before considering an ambiguity nocuous.

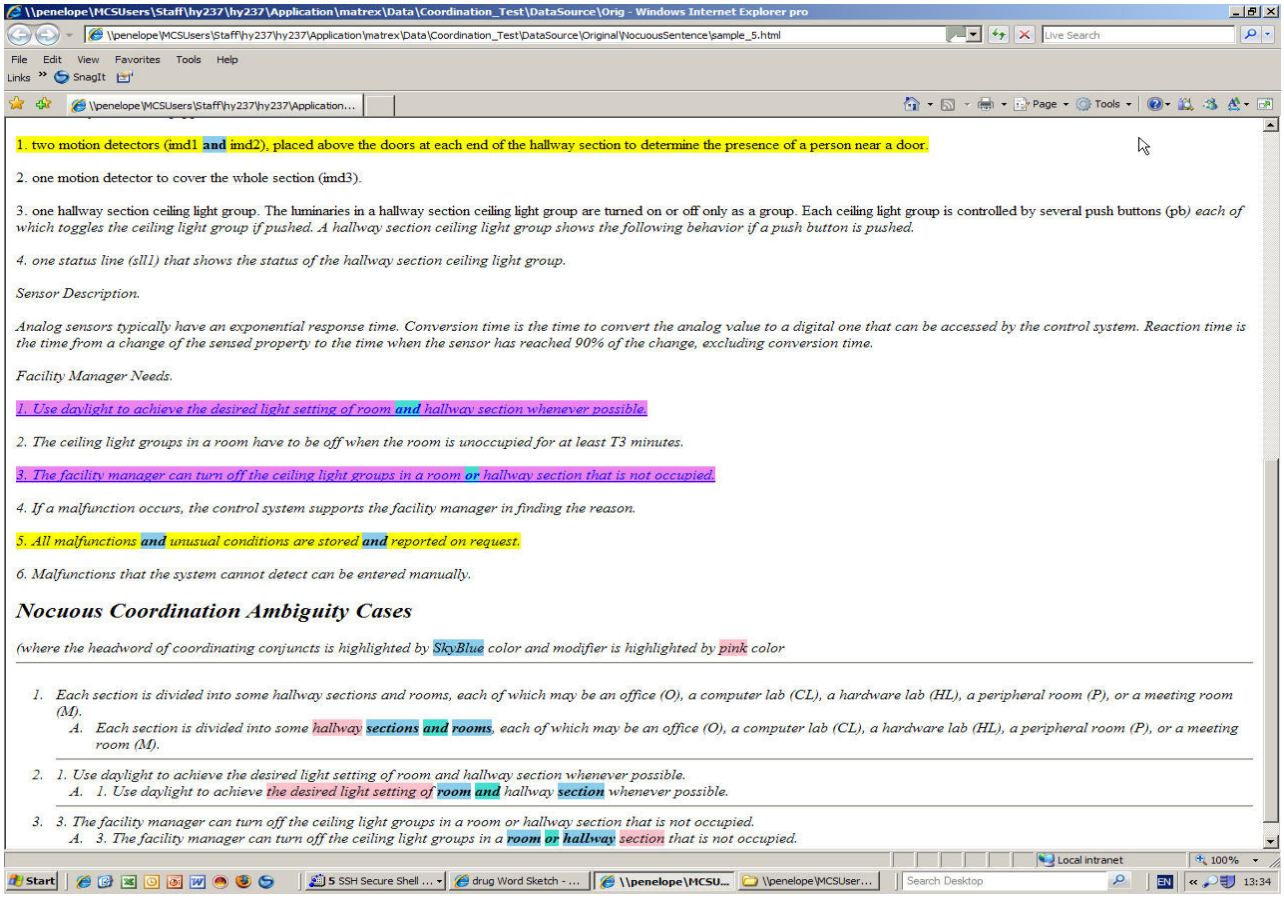

Figure 3. Sample output of the NAI tool for nocuous coordination ambiguity

Given an instance with multiple possible interpretations and a set of judgments, the *certainty* of an interpretation is calculated as the percentage of the judgments for that interpretation against the total number of the judgments for the whole instance. For instance, in Table 2, the certainty of HA for Instance (a) is 12/17=70.58%.

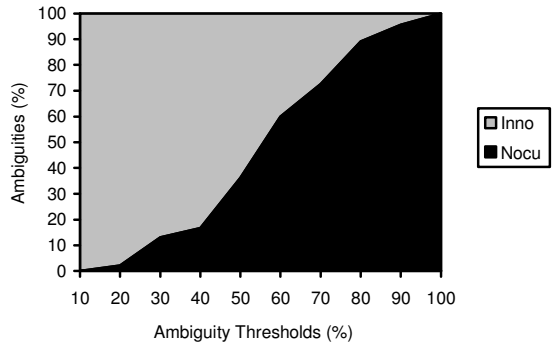

Figure 2. Proportions of interpretations at different ambiguity thresholds (Nocu=nocuous; Inno=innocuous)

Given a coordination instance, if either of the interpretations, HA or LA, has a certainty greater than the ambiguity threshold τ, we say this coordination instance displays *innocuous* ambiguity.

The relationship between ambiguity threshold and the classification of nocuous ambiguity in the dataset is illustrated in Figure 2. It shows that, at low thresholds, very few instances exhibit nocuous ambiguity, because the certainty constraint is so low that it is easier to be satisfied with a small number of agreements. However, almost all instances are classified as nocuous since it is very hard for judges to reach a consensus on the same interpretation.

Building the nocuity classifier. Our nocuity classifier was trained based on a set of heuristic scores together with the human judgments collected in the training data. More specifically, each ambiguity instance is described as a training/test instance, which is represented as an attribute-value vector, where the value of each attribute is the score of a particular heuristic described earlier. The class label of a training instance, nocuous (Y) or innocuous (N), at a given ambiguity threshold is determined by the distribution of multiple human judgments discussed earlier.

To select an appropriate machine learning (ML) algorithm to build our nocuity classifiers, we tested our dataset on a number of

ML algorithms available in the WEKA package[9] including the logistic Regression algorithm that was used in our previous work [22]. Finally, we selected the LogitBoost algorithm for building the nocuity classifier, because it performed better than other candidates including decision trees, J48, Naive Bayes, SVM, and Logistic Regression.

4.3 Applying the Classifier

To determine whether a test ambiguity instance displays nocuity or not, we presented the feature vector of the instance to the classifier, and obtained the predicted class label returned by the classifier.

Once the process of nocuous ambiguity identification was completed, we highlighted the ambiguous sentences in the original text, each of which contained at least one nocuous ambiguity. In addition, we presented the user with the extracted nocuous ambiguity instances with the identified coordination constituents highlighted by different colors. The sample output of our NAI tool for nocuous coordination ambiguity is shown in Figure 3.

5. Experiments and Results

To evaluate the performance of our tool, we used a standard 10-fold cross-validation technique in which, for each iteration, we trained on 90% data and tested on 10% of the remaining data. The performance of the system is measured in terms of precision (P), recall (R), F-measure (F), and Accuracy:

$$R = \frac{TP}{TP + FN} \qquad P = \frac{TP}{TP + FP}$$

$$F - measure = \frac{(1+\beta) * PR}{\beta^2 * P + R} \qquad Accuracy = \frac{TP + TN}{total}$$

where *TF* (true positives) is the number of correctly identified *nocuous* ambiguities, *TN* (True negative) is the number of correctly identified *innocuous* ambiguities, *FN* (false negatives) is the number of nocuous ambiguity not identified by the system, and *FP* (false positives) is the number of innocuous ambiguities which the system incorrectly classified as nocuous. The weight β is set with 0.25 in order to favor the precision. All results are averaged across ten iterations.

5.1 Performance of the Classifier

We report in this section the performance (i.e. precision, recall, and f-measure) of the ML-based classifier using the LogitBoost algorithm at different ambiguity thresholds. Figure 4 summarizes the precision and recall results of the classifier. We compared the precision of the classifier with a baseline precision (P_BL) that assumes that all of the instances are potentially nocuous ambiguities. Compared with baseline precision (P_BL), the LogitBoost classifier performed well with precision of up to 75% on average at different threshold levels. It suggests that the heuristics we developed contain distinguished features which provide strong discriminating power in determining the nocuity property of an ambiguous instance. However, at some low threshold levels, especially when the thresholds are below 45%, the classifier did not work well with respect to recall. The recall dramatically dropped down to 10%. A possible reason for this is the lack of positive cases (i.e. nocuous ambiguities) at the low threshold level (see Figure 2), which results in deterioration of performance.

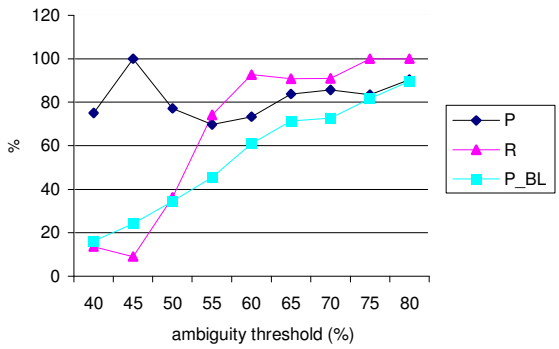

Figure 4. The performance of the classifier at different ambiguity thresholds. (P_BL=Baseline Precision)

5.2 Impact of the Heuristics

As described earlier, we introduced two more heuristics in our NAI tool: one is local-based collocation frequency which exploited the co-occurrence frequency between coordinating conjuncts and the attached modifier in the local context; the other is semantic similarity that investigated the semantic relationship between the far conjunct and near conjunct using the WordNet taxonomy. To estimate whether the two newly-added heuristics can improve system performance, we conducted another set of experiments with the heuristics only used in our previous work. We compared the F-measure performance of two sets of experiments, one is for all of the heuristics (F-All) that we described in this paper, and the other is for the heuristics except for the two newly-added heuristics (F-Part).

Moreover, to compare the performance of our proposed heuristics-based approach for nocuous ambiguity identification, we used a random model as a baseline. In the random model, we assume that each recognized ambiguity instance has the potential to be a *nocuous* ambiguity, and is counted as a positive match for the baseline model. The random model achieves an 'ideal' recall R_{BL} of 100%, and the precision and F-measure are calculated as:

$$P_{BL} = \frac{\#\ Nocuous\ \ identified\ \ by\ Judgments}{total\ \ \#\ of\ \ Ambiguitie\ s}$$

$$F - measure_{BL} = \frac{2 P_{BL} R_{BL}}{P_{BL} + R_{BL}}$$

As Figure 5 shows, compared with the baseline F-measure (F_BL), the heuristics-based approach is an effective method for the identification of nocuous ambiguity due to relatively high precision (see Figure 4). Figure 5 shows that with those selected thresholds (0.4-0.55), the heuristics-based approach exhibits a marginal improvement in F-measure compared to the baseline model. Nevertheless, the improvement gradually decreases with the increase of the threshold value. After the threshold is set to

[9] http://www.cs.waikato.ac.nz/~ml/index.html

above 0.75, the heuristics-based approach performs slightly better than the baseline model due to the quite high F-measure value.

Our results also show that the F-All performs consistently better than the F-Part throughout all of the threshold levels. It suggests that the performance of our tool did benefit from the newly-added two heuristics, which indicates that local context information and semantic relationships between the coordination constituents provide the useful clues for the identification of nocuous ambiguity.

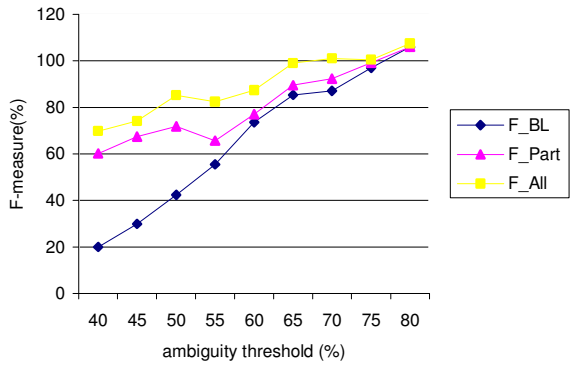

Figure 5. The Impact of the heuristics on system performance.

5.3 System Performance Comparison

In the implemented NAI tool, we chose the LogitBoost (LB) algorithm to build our ML-based nocuity classifier, which replaced the original Regression (LR) algorithm used in our previous work [22]. To evaluate the accuracy of our proposed ML-based model, we used two baseline (BL) models, BL-1 and BL-2 models. The BL-1 model assumes that all the instances are *innocuous* ambiguities, whereas the BL-2 model supposes that all the instances are *nocuous* ambiguities. As discussed previously, at low ambiguity thresholds, most of the instances are judged as innocuous ambiguities, while at high ambiguity thresholds, the majority of the instances are nocuous ambiguity (see Figure 2). Therefore, to compare fairly with the performance of the ML-based model, a good strategy is to compare it with the BL-1 model at low thresholds, and to compare it with the BL-2 model at high thresholds.

The performance comparison of the LogitBoost (LB) model against the two baseline models at different ambiguity thresholds is shown in Figure 6. The LB model performed well with an accuracy of above 75% on average at different ambiguity threshold levels. As expected, at very high and very low thresholds, the LB model did not outperform the baseline models due to the high accuracy. However, the LB model displayed its advantage when the ambiguity threshold fell in the range between 0.45 and 0.75. The LB model generally performed better than the baseline models and the maximum improvement was achieved around the 58% crossover point where the two baseline models intersect. Our tool accomplished an approximate 21% increase in accuracy. It suggested that the combined heuristics do have some capability of distinguishing nocuous from innocuous ambiguity at the weakest region of the baseline models.

Figure 6 also shows that, with the LB model, the tool improved the overall accuracy with an increase of approximate 4.4% on average compared with the previous model, the Logistic Regression (LR) model. The possible explanation is that the LB algorithm is more suitable for the dataset with the numeric-attribute feature vectors.

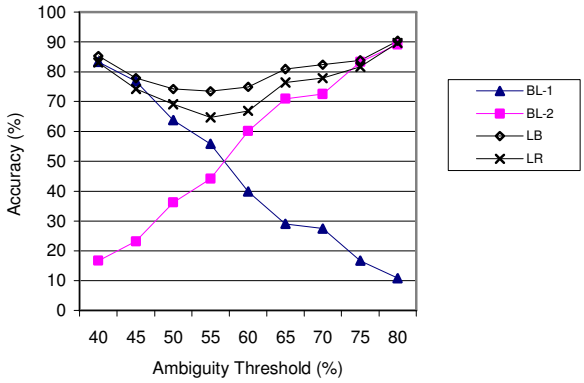

Figure 6. The performance comparison of the ML-based model to the two baselines, BL-1 and BL-2. (BL-1=baseline (all innocuous); BL-2=baseline (all nocuous); LB=LogitBoost; LR=Logistic Regression))

6. Related Work

NLP tools applied to NL requirements. A number of natural language processing (NLP) systems or tools applied to NL requirements had been developed in recent years. Ambriola and Gervasi [2] developed a web-based NLP tool, called Circe, which was designed to facilitate the gathering, elicitation, selection, and validation of NL requirements. IBM Rational Doors[10], a requirements management tool, provides relevant functional modules for the generation of NL requirements and the traceability among NL requirements. Goldin and Berry [11] implemented a tool called Abstfinder to identify the abstractions from natural language text used for requirements elicitation. Lee and Bryant [16] developed an automated system to assist the engineers to build a formal representation from informal requirements like NL requirements.

Ambiguity in NL requirements. Several studies dealing with ambiguity identification have aimed to help improve the quality of NL requirements documents. Some tools have been developed specifically to detect, measure, or reduce possible structural ambiguities in text. Kamsties et al. [12] describe pattern-driven inspection technique to detect ambiguities in NL requirements. Fuchs and Swhwitter [9] present a restricted NL, called Attempt Controlled English (ACE), to translate specifications into sentences in first-order logic in order to reduce the ambiguity in requirement specifications. Mich and Garigliano [17] investigate the use of a set of ambiguity indices for the measurement in syntactic and semantic ambiguity, which is implemented using an NLP system called LOLITA. Kiyavitskaya et al. [15] proposed a two-step approach in which a set of lexical and syntactic ambiguity measures are firstly applied to ambiguity identification, and then a tool to measure what is potentially ambiguous specific to each sentence.

[10] http://www-01.ibm.com/software/awdtools/doors/

Finally, some of the tools have been developed to examine the quality evaluation of requirements. These include QuARS [7], ARM [23], and the tool by Fantechi et al. [8] for use case requirements. These approaches define a quality model composed of a set of quality metrics (e.g., vagueness, subjectivity, optionality, weakness, etc.), and develop analysis techniques based on a linguistic approach to detect the defects related to the inherent ambiguity in the requirements.

Coordination Ambiguity. In last decade, a number of approaches have been proposed to address ambiguity resolution in coordination constructions. Previous research efforts have generally focused on either corpus-based statistical methods (e.g., co-occurrence frequency between coordinating conjuncts and the modifier over a text corpus such as web resource [18] or Wall Street Journal [10]), or linguistic approaches that made use of part-of-speech (POS) tagging and shallow (e.g., phrase) and deep parsing (e.g., parsing tree) information to apply pattern- or rule-based matching [1, 19, 21]. In addition, similar to our work, Resnik [20] took advantage of semantic similarity of a taxonomy to resolve coordination ambiguity involving nominal compounds.

Unlike other related ambiguity work that attempts to resolve ambiguity by applying disambiguation techniques to select the most appropriate reading, our studies present readers with nocuous ambiguities which may potentially be misunderstood by different readers, and allow these readers to determine the preferred interpretations.

7. Conclusions and Future Work

Natural language still prevails in a large number of requirements documents. We need ways to cope with the ambiguity inherent in natural language. It is important to develop scalable automated techniques to detect potential nocuous ambiguities in natural language requirements, in order to minimize their side effects at the early stages of the software development lifecycle.

In this paper, we described an automated approach to characterize and identify potentially nocuous ambiguities, which have a high risk of misunderstanding among different readers. Given a natural language requirements document, ambiguous instances contained in the sentences were first extracted. Then, a machine learning approach was employed to classify ambiguous instances as nocuous or innocuous, subject to a given ambiguity threshold. A prototype tool, which focuses on coordination ambiguity, was implemented in order to illustrate and evaluate our approach. We reported on a set of experimental results to evaluate the performance and effectiveness of our automated approach. The results show that our tool is capable of accurately detecting potentially dangerous ambiguities in the NL requirements.

Based on significant technical development and substantive empirical studies, we believe that the application of our approach is lightweight and usable in that it allows requirements analysts to experiment iteratively to identify potential nocuous ambiguity in requirements, depending on their chosen analysis sensitivity threshold.

However, a number of interesting issues remain to be investigated in order to improve our tool's performance and validate its use in practice. First, more heuristics need to be developed to further explore aspects of ambiguity that enhance the accuracy of our tool. Second, our current tool is specific to coordination ambiguity. It is necessary to extend it to a wider range of ambiguity types, for example, to other types of structural ambiguity like prepositional attachment ambiguity and semantic ambiguities such as anaphora. We have begun to explore the identification of nocuous ambiguity in terms of anaphora ambiguity in our ongoing work [24]. Third, we need and plan to make our tool more widely accessible to validate its use in practice. This may include migrating the technology to a web-based environment, providing an automated analysis as some kind of web service. Indeed, we envisage that this automated support for ambiguity analysis should fit into a number of requirements management environments, in which requirements authors are able to invoke such analysis tools in the same way as writers invoke spell checkers. We are currently investigating the development of this capability within a well-known commercial tool.

8. ACKNOWLEDGMENTS

This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) as part of the MaTREx project (EP/F068859/1), and by the Science Foundation Ireland (SFI grant 03/CE2/I303_1). We are grateful to our research partners at Lancaster University for their input, and to Ian Alexander for his practical insights and guidance.

9. REFERENCES

[1] Agarwal, R., and Boggess, L. 1992. A simple but useful approach to conjunct identification. In Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics, 15–21.

[2] Ambriola, V., and Gervasi, V. 1997. Processing natural language requirements. In Proceedings of the 12th international conference on Automated software engineering 36-45.

[3] Berry, D. M., Kamsties, E., and Krieger, M. M. 2003. From contract drafting to software specification: Linguistic sources of ambiguity.

[4] Boyd, S., Zowghi, D., and Farroukh, A. 2005. Measuring the expressiveness of a constrained natural language: An empirical study. In Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05), Washington, DC, 339-352.

[5] Brill, E., and Resnik, P. 1994. A rule-based approach to prepositional phrase attachment disambiguation. In Proceedings of COLING, 1198 - 1204.

[6] Chantree, F., Nuseibeh, B., De Roeck, A., and Willis, A. 2006. Identifying Nocuous Ambiguities in Natural Language Requirements. In Proceedings of 14th IEEE International Requirements Engineering Conference (RE'06), Minneapolis, USA, 59-68.

[7] Fabbrini, F., Fusani, M., Gnesi, S., and Lami, G. 2001. The linguistic approach to the natural language requirements, quality: benefits of the use of an automatic tool. In Proceedings of the twenty sixth annual IEEE computer society-NASA GSFC software engineering workshop, 97–105.

[8] Fantechi, A., Gnesi, S., Lami, G., and Maccari, A. 2003. Applications of Linguistic Techniques for Use Case Analysis. Requirements Engineering, 8(9), 161-170.

[9] Fuchs, N. E., and Schwitter, R. 1995. Specifying logic programs in controlled natural language. In Proceedings of the Workshop on Computational Logic for Natural Language Processing, 3–5.

[10] Goldberg, M. 1999. An unsupervised model for statistically determining coordinate phrase attachment. In Proceedings of ACL, 610-614.

[11] Goldin, L., and Berry, D. M. 1994. Abstfinder, a prototype abstraction finder for natural language text for use in requirements elicitation: design, methodology, and evaluation. In Proceedings of the First International Conference on Requirements Engineering, 18–22.

[12] Kamsties, E., Berry, D., and Paech, B. 2001. Detecting ambiguities in requirements documents using inspections. In Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01), 68-80.

[13] Kilgarriff, A. 2003. Thesauruses for natural language processing. In Proceedings of NLP-KE, 5–13.

[14] Kilgarriff, A., Rychly, P., Smrz, P., and Tugwell, D. 2004. The Sketch Engine. In Proceedings of the Eleventh European Association for Lexicography (EURALEX), 105–116.

[15] Kiyavitskaya, N., Zeni, N., Mich, L., and Berry, D. M. 2008. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements Engineering Journal 13, 207–240.

[16] Lee, B. S., and Bryant, B. R. 2004. Automation of software system development using natural language processing and two-level grammar. Radical Innovations of Software and Systems Engineering in the Future. Springer, Heidelberg 219–233.

[17] Mich, L., and Garigliano, R. 2000. Ambiguity measures in requirement engineering. In Proceedings of international conference on software—theory and practice (ICS2000), 39–48.

[18] Nakov, P., and Hearst, M. 2005. Using the Web as an Implicit Training Set: Application to Structural Ambiguity Resolution. In Proceedings of HLT-NAACL'05, 835–842.

[19] Okumura, A., and Muraki, K. 1994. Symmetric pattern matching analysis for English coordinate structures. In Proceedings of the 4th Conference on Applied Natural Language Processing, 41–46.

[20] Resnik, P. 1999. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. Journal of Artificial Intelligence Research (JAIR), 11, 95–130.

[21] Rus, V., Moldovan, D., and Bolohan, O. 2002. Bracketing compound nouns for logic form derivation. In Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS), 198 – 202.

[22] Willis, A., Chantree, F., and De Roeck, A. 2008. Automatic Identification of Nocuous Ambiguity. Research on Language & Computation, 6(3-4), 1-23.

[23] Wilson, W. M., Rosenberg, L. H., and Hyatt, L. E. 1997. Automated analysis of requirement specifications. In Proceedings of the Nineteenth International Conference on Software Engineering (ICSE), 161–171.

[24] Yang, H., De Roeck, A., Gervasi, Vincenzo, Willis, A., and Nuseibeh, B. 2010. Extending Nocuous Ambiguity Analysis for Anaphora in Natural Language Requirements. In Proceedings of the 18th IEEE International Requirements Engineering Conference (RE'10) (In Press).

[25] Yang, H., De Roeck, A., Willis, A., and Nuseibeh, B. 2010. A Methodology for Automatic Identification of Nocuous Ambiguity. In Proceedings of the 23rd International Conference on Computational Linguistics (Coling'10) (In Press).