Processing Natural Language Software Requirement Specifications

Miles Osborne*and C K MacNish†

Department of Computer Science, University of York,
Heslington, York YO1 5DD, UK
{miles,craig}@minster.york.ac.uk

Abstract

Ambiguity in requirement specifications causes numerous problems; for example in defining customer/supplier contracts, ensuring the integrity of safety-critical systems, and analysing the implications of system change requests. A direct appeal to formal specification has not solved these problems, partly because of the restrictiveness and lack of habitability of formal languages. An alternative approach, described in this paper, is to use natural language processing (NLP) techniques to aid the development of formal descriptions from requirements expressed in controlled natural language. While many problems in NLP remain unsolved, we show that suitable extensions to existing tools provide a useful platform for detecting and resolving ambiguities. Our system is demonstrated through a case-study on a simple requirements specification.

1. Introduction

System Requirement Specifications (SRSs) often form the basis of the contract between the customer and the system supplier. Consequently, they need to be precise to serve as the basis of a specification of what the system is supposed to do. They also need to be clear enough for customers to confirm that a system built from the SRS satisfies their needs. This need for precision and clarity places demands upon the particular requirement specification technique used to create the SRS. For example, SRSs expressed solely in natural language may be comprehensible by the customer, but an SRS expressed in this way will be ambiguous, possibly inconsistent, and probably unmanageably large. However, SRSs expressed in terms of formal techniques such as finite state machines (for example [19]), program description languages ([3]), or PAISLey ([20]) may be clear to the system designers, but will be difficult to comprehend by non-computer specialists. One way to solve this trade-off between the need for clarity and the need for precision is to use a formal method for structuring, and natural language as elaboration of this structuring. In this paper we concentrate upon the natural language aspects of a SRS and have little to say about formal approaches to SRS construction.[1]

Unfortunately, using natural language to construct SRSs is still problematic, even when combined with formal techniques. Perhaps the main problem is that natural language is ambiguous, with many possible interpretations. An ambiguous SRS may lead to a system being built that does not meet the customer's requirements. Another problem is that a SRS expressed in natural language may be vague. By *vagueness*, we mean a lack of clarity. Again, vagueness may lead to changes in the delivered system. Both of these problems can be dealt with by proof-reading the natural language components of the SRS and, when necessary, augmenting informally expressed requirements with a corresponding formal statement eliminating some of the unintended meanings. The result of this process would be a clearer SRS, and one less likely to contain errors. Whilst proof-reading and augmentation can be performed manually, a better solution, which will be less burdensome, and possibly more thorough, would be to use a semi-automated editor, along with a translator of natural language sentences into a formal representation. Techniques drawn from natural language processing (NLP) can be used for both of these tasks. In this paper we present a set of NLP techniques that can be used to enhance the quality of SRSs. Note that

*Current address: Department of Computing Science, University of Aberdeen, Aberdeen AB9 2UE, Scotland

†Current address: Department of Computer Science, University of Western Australia, Nedlands W.A. 6907, Australia

[1] Experience with a range of industries suggests that approximately 90% of SRSs are expressed solely in natural language, less than 10% use both formal methods and natural language and less than 1% use just formal methods (John McDermid, personal communication).

Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)
0-8186-7252-8/96 $10.00 © 1996 IEEE

here we principally concentrate upon dealing with ambiguity. Later work will deal with vagueness.

The structure of this paper is as follows. Section 2 presents some of the problems associated with processing the natural language aspects of SRSs, and discusses a solution to the problem of computationally treating this language use. Section 3 then presents our implementation of this solution. In section 4 we give an overview of the framework that we use for structuring SRSs, along with an example SRS in this notation. The example SRS is then processed, using our system, in section 4.3. Finally, section 5 concludes the paper with a discussion of future work.

2. NLP and SRSs

NLP systems typically consist of a *grammar* describing the syntax of a natural language, a *lexicon* describing lexical information about words, a semantic component that is used to construct the literal meaning of sentences, and a pragmatic component which is used to construct the non-literal meaning of sentences. They also contain a *parser*, which is a program that produces phrase structure trees for sentences in the language defined by the grammar.

Unfortunately, there are numerous problems associated with processing unrestricted natural language, as found in SRSs.[2] For example:

- The lexicon may fail to contain entries for all of the words that the system might encounter, or, the lexicon might not contain all of the word senses that a word might have. In the first case, the NLP system would simply fail to process the sentence containing the unknown word. In the second case, the system might either fail to process the sentence, or might fail to assign the correct analysis to the sentence.

- The grammar might assign more than one parse to a sentence. These parses might reflect genuine ambiguity, or they might be spurious. In both cases the system will need to select one or more of these parses and discard the others, otherwise there is a danger of the system becoming swamped with competing analyses.

- The semantics might fail to account for all of the constructs that the system parsed. For example, to date there is no adequate treatment of tense. This theory incompleteness (with respect to the syntax) is clearly a limiting factor.

[2]By unrestricted, we mean language that is used to communicate between humans, without regard to the needs of machine.

All of these are hard problems and undermine the use of NLP in requirements analysis. The impact of these problems can be reduced, however, by restricting the scope of the language that is used [12]. That is, only allow a certain set of words in the lexicon, only allow a certain set of syntactic constructs, only allow a certain semantic theory to be used to interpret the sentences, and so on. A language that is restricted in this way is known as a *controlled language* (CL).

CLs have been used in a variety of applications. For example, the aircraft industry has developed a CL (the International Aerospace Maintenance Language [1]) designed for use by maintenance staff who are not necessarily native English speakers; this standard has been used by Boeing since 1990 [10]. The Caterpillar Tractor Company has been designing and adapting CLs for the last 15 years. Indeed, researchers at the University of Washington and Boeing have carried out experiments suggesting that documents written in a CL are more comprehensible than documents written in an unrestricted language [16].

Using a CL for requirements analysis is not necessarily straightforward:

- Controlling a natural language reduces the habitability of the system. That is, the CL might be so restricted that it becomes irritating to use and arguably becomes a formal, unnatural language. For such cases, the user might be better off actually using a formal language. However, trying to reduce the gap between the CL and unrestricted natural language re-introduces the above problems.

- The user needs guidance on how to phrase requirements in terms of the CL. This means that when the user deviates from the CL, appropriate information is given, enabling linguistically naïve users to re-phrase their statement in terms of the CL. Also, it is often difficult for humans to determine why a sentence is ambiguous, and what the alternative readings mean. Hence, the system needs to present the alternatives in a perspicuous manner.

- Natural languages are not always an appropriate medium for expressing all requirements. For example, algorithms are better expressed in some sort of task-related formalism; structural relationships are better expressed in terms of diagrams, and so on.

In section 3 we describe how we address the first two aspects of using a CL for requirements analysis. We realise that natural language is not suitable for all aspects of a SRS. As such, in section 4 we outline the formal, structured aspect of the framework we also use for constructing a SRS.

3. The Newspeak System

In this section we present our choice of CL and a set of practical additions necessary before we can make use of the CL. The CL and additions constitute the *Newspeak* System. We end with an example of our system in action.

3.1. Choice of Controlled Language

In the previous section, we commented that CLs might be unusable if they are overly restricted. We address this problem by starting with a CL that covers as much of the language used to construct SRSs as possible. We can then tailor the CL to the sublanguage of SRSs. Users of such a system will not have to spend much time learning to use the CL, and instead can concentrate upon the task of creating a readable SRS. Given this desire for habitability, we have chosen as the basis of our CL the Alvey Natural Language Toolkit (ANLT) [4].

The ANLT contains one of the broadest covering grammars in existence, a lexicon of 40,000 entries (based upon the Longman's Dictionary of Contemporary English), a semantic component that assigns one or more logical forms (LFs) to each sentence parsed, and an optimised parser capable of using the wide-covering grammar for parsing naturally-occurring sentences. As an indication of the coverage of the grammar, we have used an extended version of the ANLT to parse 78% of 200 sentences taken from the Dynix Automated Library Systems Searching Manual [14].

3.2. Practical additions to the ANLT

Selecting a CL is only part of the task of designing an appropriate system. We also need to provide feedback to help the user create sentences that are within the CL, determine what are the ambiguities present in sentences they write, and also to enable confirmation that the logical form of a sentence is close to what the user intended originally. The ANLT does contain facilities for viewing sentences, parse trees, and LFs. However, these presuppose a training in computational linguistics. We have therefore extended the ANLT in the following ways:

- The ANLT's parser produces all possible parses for an ambiguous sentence. Some of these parses will be contextually appropriate for the sentence, whilst others will be contextually inappropriate (spurious). Also, these parses are unordered, and may number in the hundreds, if not thousands. Hence, we have added a *parse selection mechanism* to rank the parses in terms of plausibility.
- When the parser cannot parse a sentence, it gives no indication of what the reason for this failure might be. We have therefore added an *error diagnostic* facility.
- We are investigating ways of *presenting ambiguity* without recourse to syntactic definitions. These are additional to the ANLT's explanatory facilities.
- Finally, we have added a mechanism to place *resource bounds* upon the parser. These help ensure that the parser does not spend too much time parsing a sentence.

We now briefly describe these extensions to the ANLT.

3.2.1 Parse selection

A *parse selection mechanism* is a decision procedure that can be used to rank the parses. The ranking is intended to reflect how close some parse is to the 'true' parse for some sentence. We would like to give parses that are likely to be spurious a low ranking, whilst those that are likely to be appropriate a higher ranking. It then becomes a matter of presenting to the engineer firstly an indication of how many parses some sentence has, and then the first m parses for that sentence. Clearly, selecting a parse out of these m parses is far easier than selecting a parse from the entire set of parses.

We have implemented a simple approach as follows. Firstly, we trained the mechanism by parsing approximately 750 sentences that were supplied with the ANLT. For each sentence we told the parser which was the preferred parse out of those parses produced for that sentence, recorded which rules were used in that parse, and noted the frequency of each rule's application in the parses. We then normalised the frequencies against the total number of rule applications. Now, during parsing, we score parse trees by taking the product[3] of the score of the rules used in each tree. For rules used in the tree that were not encountered in the training set of parses, we assign a small value. To avoid underflow, we use logarithms in place of the normalised scores, and simply sum the logarithms. The

[3]We take a product to reflect the intuition that roughly speaking, the rules used in a parse are mutually independent.

trees are then sorted by their score, and the first m ($m = 5$) parse trees are processed and presented to the engineer. Note that this parse selection mechanism is very simple, and better approaches exist (for example [2]). In general, better approaches make use of context, which we ignore. We intend to investigate these alternatives in later work.

3.2.2 Error diagnosis

For error diagnosis, we try to determine the circumstances under which an unparsable sentence might be parsed. This is equivalent to finding which of the partial parses produced for that sentence can be repaired and combined together to form a parse for the entire sentence. If these circumstances can be discovered, then a meaningful error message can be presented to the engineer.

An unparsable sentence will in general have many possible partial parses that could be completed. This is especially so when using a wide-covering grammar. Trying to determine which of these partial parses corresponds to a broken parse for the sentence, and which are simply spurious, is extremely difficult. If the distinction cannot be made, then naïvely basing error reports about all of these partial parses will quickly swamp the engineer with a cascade of spurious messages. Instead of being exhaustive, we take a heuristic approach and search for the partial parse that accounts for most of the sentence and assume that that is what the engineer intended. Next, we look for partial parses to the left and to the right of this maximal parse fragment. If either of these adjacent fragments can be found, we then attempt a series of parsing actions to try and join them to the central parse fragment. Each parsing action corresponds to a *relaxation* of some grammatical constraint. For example, if the system encounters a number disagreement (such as *the brakes is applied*), it will try to relax the constructions that enforce number agreement. Relaxation is not new (see, e. g. [6, 8, 9, 17, 18]) and has been used to deal with ill-formed sentences, and to generate error reports for such sentences. We use a form of *default unification* to implement relaxation [14].

3.2.3 Presentation of ambiguity

Regarding the presentation of ambiguity to the user, currently we simply display a bracketed parse of the sentence. We do not present notions such as 'Noun Phrase' and the like. Whilst this helps to spot plausible parses from implausible parses, it fails to show non-structural differences between parse trees (for example tense).

3.2.4 Resource bounds

Finally, we place a resource bound upon the parser: it will halt parsing when x ($x = 25000$) *edges* have been constructed. An edge is a data structure that the parser uses to construct parse trees. From a practical perspective, this bound helps the parser from growing too large (and so start to thrash, or eventually crash). From a more intuitive perspective, this reflects the idea that if a sentence has a parse, then that parse can be found relatively quickly. Otherwise, the extra effort is simply wasted work. Previous authors have used resource bounds in a similar way [13, 15]. Note that our use of resource bounds only makes sense for parsers, such as the ANLT's chart parser, which have a form of 'best-first' search.

3.3. An example

To illustrate these facilities, we give a small example of our system in action. A larger example is shown in section 4.

Suppose the analyst submitted the ill-formed sentence:

1 *A brake are applied*

to the system. The system does not generate this sentence, and informs the user that there is a number disagreement between *a brake* and *are applied*:

```
155 Parse#(> a brake are applied
Max. spanning fragment: (a brake)
Plu disagreement between: (a brake) and
(are applied)
```

Changing the sentence to:

2 *A brake is applied*

produces the following result:

```
Parse#(> >) a brake is applied
1 parse trees in total.
Tree 0 has a score of -29:
(((a ((brake))) (is (((apply +ed)))))
```

Here, we show the bracketed parse for the sentence, and the score that the system has assigned for that parse. In general, sentences have more than one parse, and the scores give an indication of how plausible some parse is. The system also generates a LF for the sentence:

some(x1), entity(x1)
some(x2), sg(x2) ∧ brake(x2)
some(e1), apply(pres(e1), x1, x2)

As can be seen, the LF is an expression in first-order predicate logic, augmented with 'event' variables, and generalised quantifiers. For the purposes of this paper, these details are unimportant. However, the LF can be paraphrased as 'there is some entity x1, and some singular brake x2, such that at some present event e1, x1 applies the brake'.

4. Case study: using Newspeak to refine the requirements for a Landing Control System

In this section we demonstrate, by way of a case-study, the use of the Newspeak system to clarify the expression of requirements. To provide some context for reading the example, we begin with a brief description of our framework for structuring requirements. Due to space restrictions we then present the example, a specification for a simplified landing control system, with little additional explanation. Further details of both the framework and the example and its motivation are contained in [7].

The main task in this section is then to successively take the assertions (or statements about the design) and analyse these using the Newspeak system. While the specification is a simple one, it was constructed for another purpose (a logical analysis), without automatic parsing in mind, so the sentences are independent of the analysis methods used. We show that an educated user, in conjunction with the tool, is able to refine the sentences into a form which the tool is able to parse and generate a logical semantics.

4.1. Goal-Structure Framework

We use a framework called *goal-structured analysis* (GSA) [7] as the basis for constructing SRSs. The motivation for this framework is that, in order to analyse the effects of changing particular requirements, we need to know the semantic relationships between requirement statements, or *assertions.* This is achieved by structuring the specification according to system goals (see, for example, [5]), recording appropriate information about design decisions, and associating logical statements with assertions. One of the aims of the Newspeak system is to generate these logical statements.

In brief, a goal structure consists of a directed, acyclic graph of structured objects. Each object is a set of slots. Each slot may contain an informal, and a corresponding formal component. The informal component allows the developers to describe the system in natural or semi-structured language. The formal component is used to guarantee properties of the system and to permit automated analysis. Note that both components need not necessarily be filled by the same engineer — for example the first may be the responsibility of the system designer while the second may contain a formalisation of this information by a specialist in formal languages.

Each object is either a *goal* (a decomposable property of the system that we would like to achieve), an *effect* (similar to a goal, but not necessarily desirable), a *fact* (a "bottom-level" property which will not be decomposed further), or a *condition* (similar to a fact, but only holding in particular scenarios). A natural representation for goals, effects, facts, conditions and their associated information is a *frame*, a data structure commonly used for representing hierarchical information in AI (see for example [11]).

4.2. The Landing Control System

We now provide an example SRS expressed in terms of Goal Structuring. The motivation for this example is to provide material for subsequent refinement in section 4.3. Although the example is somewhat contrived (see [7] for further details), it does serve as material for our system. In particular, the SRS was written without regard to it being analysed by our system. Hence, some of the problems associated with real-world SRSs will be present in our SRS.

The example is based upon a simplified landing control system for an aircraft. Briefly, the example involves the development of an automated system for applying wheel brakes and reverse thrusters with the aim of stopping an aircraft within 1000m of landing.

The initial SRS for the landing control system consists of the following goals, effects, facts, and conditions:[4]

Fact	
Label	F1
System	Aircraft dynamics
Assertion	The aircraft will stop within 1000m if wheel brakes are applied when speed $\leq$ 154km/h.
Rationale	Engineering tests

Fact	
Label	F2
System	Aircraft dynamics
Assertion	The aircraft will stop within 1000m if wheel brakes and reverse thrusters are applied when speed $>$ 154km/h and $\leq$ 170km/h.
Rationale	Engineering tests

[4]To conserve space, we present all the slots for facts F1 and F2, just the assertions for fact F3 and goals G2 and G3, and omit all other goals, facts, effects and conditions that were presented in the original paper [7]. A fuller version of this paper, describing what we have omitted, is available from the authors.

Goal	
Label	G1
Assertion	Stop aircraft within 1000m

Goal	
Label	G2
Assertion	Apply reverse thrust

Fact	
Label	F3
Assertion	If wheel loads > 12 tonnes then reverse thrust is applied

Goal	
Label	G3
Assertion	Apply wheel brakes

4.3. The Refinement Process

We can now demonstrate the use of our system in successively refining the Landing Control system. To be precise, we shall process all of the informal assertions made in each frame, detect ambiguities, and then translate each informally specified assertion into a logical form. For the sake of brevity, and since it is the most important slot for change analysis, we will focus on the assertions. The same refinement process can, of course, be applied to other slots.

Starting with the assertion in F1:

3 *The aircraft will stop within one thousand meters if wheel brakes are applied when speed is less than or equal to one hundred and fifty four kilometres per hour*

This sentence receives six, equally ranked syntactic analyses. The parses can be divided into two groups (each group in turn being three ways ambiguous). The first ambiguity is whether the phrase *when speed is less than or equal to one hundred and fifty four kilomeres per hour* modifies the phrase *applied* or the phrase *if wheel brakes are applied.* The second ambiguity is whether the phrase *the aircraft* is singular, plural, or neither. In general, each ambiguity multiplies the number of parses. If we decide that only a single aircraft will stop:

4 *An aircraft will stop within one thousand meters if wheel brakes are applied when speed is less than or equal to one hundred and fifty four kilometres per hour*

we produce a sentence with two parses. Selecting the first parse (which, although omitted, is preferable), produces five distinct logical forms. Some of these logical forms differ from each other according to various interpretations of the phrase *speed.* One interpretation is that of a particular speed (say that of the aircraft). Another interpretation is of all speeds (for example, the speed of tractors, grass growing, and so on). We can eliminate this semantic ambiguity by changing the phrase *speed* to *its speed.* Making this change results in two parses, as before. Selecting the first parse results in two distinct LFs. These LFs differ from each other regarding the interpretation of the phrase *wheel-brakes.* This is the same sort of difference as we saw with the previous example of semantic ambiguity. Hence, using the same approach as before, we change *wheel-brakes* to *its wheel-brakes.* This also results in two parses; the first parse, when selected, has only a single interpretation, as shown in figure 1. For future sentences, we

```
some(x1), sg(x1) ∧ aircraft(x1)
1000(x2), indef(x2) ∧ pl(x2) ∧ metre(x2)
some (x3), entity(x3)
some(x4), the(y1), (sg(y1) ∨ ms(y1)) ∧ nonhuman(y1)
pl(x4) ∧ possessed-by(x4, y1) ∧ (wheel-brake x4)
some(x5), the(y2), (sg(y2) ∨ ms(y2)) ∧ nonhuman(y2)
sg(x5) ∧ possessed-by(x5, y2) ∧ speed(x5)
154(x6), indef(x6) ∧ pl(x6) ∧ kmh(x6)
some(e1), stop (fut(e1), x1) ∧ in (fut(e1),x2)
some(e2), apply(pres (e2), x3, x4)
some (e3), be(pres(e3), leq-to( x5, x6))
when(pres(e2),pres(e3))
fut(e1) → pres(e2)
```

Figure 1. A logical form for sentence 4

shall omit the LF. This is for the sake of brevity. The final sentence is:[5]

5 *An aircraft will stop within 1000m if its wheel brakes are applied when its speed is* $\leq$ *154 kmh.*

Turning to the assertion in F2, the sentence:

6 *The aircraft will stop within 1000m if wheel brakes and reverse thrusters are applied when speed* $>$ *154 kmh and* $\leq$ *170kmh*

can be immediately re-written into the sentence:

7 *An aircraft will stop within 1000m if its wheel brakes and reverse thrusters are applied when its speed* $>$ *154 kmh and* $\leq$ *170kmh*

This gives five parses. These parses differ from each other in terms of what is conjoined with what. We could eliminate these ambiguities by re-writing the sentence into a set of sentences. Instead, we keep the sentence as it is, and simply associate with it the intended, disambiguating, LF.

Regarding the assertion in G1, the sentence:

8 *Stop aircraft within 1000m*

receives four parses. Some of these ambiguities concern the plurality of *aircraft.* Changing the sentence to:

9 *Stop an aircraft within 1000m*

[5]Starting with this sentence, we use conventional mathematical notation in our presentation. This is for legibility.

reduces the number of parses to two. The first parse reflects the reading 'stop an aircraft that is within one thousand meters'; the second parse the reading 'there is an aircraft that will stop and this aircraft will be stopped within one thousand meters'. Of the two interpretations, the second reading is preferred.

One problem with the system is that it does not suggest ways of enriching the sentence so as to make it clearer. For example, the sentence says nothing about the conditions under which an aircraft will stop. In lieu of a theory of what makes a good requirement specification, it is up to the engineer to realise this deficiency. Suppose that G1 is intended to mean that the aircraft should stop after landing:

10 *After touchdown stop an aircraft within 1000m*

Such a sentence receives two parses (as before). Selecting the second reading gives a single logical form.

Regarding the assertion in G2, the sentence:

11 *Apply reverse thrust*

does not receive a parse. The reason for this failure is that the sentence lacks a determiner.[6] Unfortunately the system does not diagnose missing words and gives no help for this example. However, it is not too difficult to see that the sentence should be changed to:

12 *Apply the reverse thrust*

This sentence receives a single reading and a single logical form. As with the assertion in G1, this sentence ought to be made more specific. Hence, we change the assertion to:

13 *After touchdown apply the reverse thrust*

This again receives one parse, and a single logical form.

Turning to the assertion in F3, the sentence:

14 *If wheel loads > 12 tonnes then reverse thrust is applied*

does not receive a parse. The reason for this failure is that the grammar lacks a rule to generate sentences of the form *if ... then ...* . The grammar can generate sentences that omit the *then* and indeed does generate the assertion, when re-written without the *then*, giving it 3 parses. Only the first parse is worth considering and the other, less highly ranked parses are spurious. This selected parse is semantically ambiguous, with five distinct readings. As with previous, similar sentences, these ambiguities revolve around *thrust reverse* and *wheel loads* being underspecified. Rephrasing the sentence as:

[6]Interestingly enough, sentence 11 could be interpreted as an imperative statement. Failing to generate this sentence illustrates the issue of habitability.

15 *if the wheel loads > 12 tonnes apply the reverse thruster*

produces two parses. The highest ranked parse is the one we want. Selecting this parse produces a single logical form. Note that the system has not detected the deviation in the SRS from *reverse thrust* to *reverse thruster.*

The assertion in G3 is treated similarly to the assertion in G2, becoming:

16 *After touchdown apply the wheel brakes*

This has a single logical form.

In conclusion, we have processed the assertions in the SRS, detected ambiguities, and produced corresponding LFs for each informally stated assertion. Although we make no claim regarding the legibility of the resulting SRS, we believe that it is now more rigorous in terms of the allowable interpretations, and therefore clearer.

5. Discussion

In this paper we have discussed how NLP can support the RE process, and highlighted some of the problems associated with providing this support. We have presented a system that detects ambiguities in SRSs, and if the sentence is parsed, produces an associated logical form. A demonstration of our system showed typical syntactic and semantic ambiguities being detected, and resolved. One consequence of using a system such as ours is that it forces the requirements analyst to focus much more upon what is actually being written. We believe that this attention to detail will result in clearer SRSs, with a greater chance of meeting the customer's requirements. As should be evident, using the system requires skill on the part of the user, with an understanding of what the SRS is supposed to mean. Our tool supports, and does not remove, the need for a skilled requirements analyst.

Apart from creating clearer SRSs, another aspect of our work is concerned with automated reasoning about SRS. We are looking at using the LFs created to detect inconsistencies in the SRS. This would aid in validating the SRS. Also, we are investigating the use of LFs to manage change in requirements.

There is still much work that needs to be carried out before our system can be used in the field. For example, there is no use of a data dictionary. Clearly, use of a data dictionary to help specify the lexicon would ensure that terms are used consistently.

Related to the previous point, there is no provision for dealing with lexical incompleteness.

There is no pragmatic component. As it stands, the system cannot resolve (for example) anaphoric references, or ensure that style guidelines are adhered to.

The error reporting capability is modest, and does not address (for example) missing words.

Often, it is difficult to determine how parses differ, and what these differences mean. Hence, the system needs a generation component to paraphrase the logical forms in terms accessible to the engineer.

We also need to carry-out realistically large case studies in order to evaluate our work. To this end we are working on case studies with Rolls Royce and British Aerospace.

Although there is a great deal more work which can be done to improve such a system, we believe, as demonstrated by our detection of ambiguity in SRSs, that NLP does have a valuable role to play in requirements engineering.

Acknowledgements

The authors would like to thank David Duffy, John McDermid and Jonathan Moffett for comments, This work was supported by the DTI/EPSRC Proteus Project IED4/1/9304.

References

[1] AECMA. AECMA/AIA Simplified English: a Guide for the Preparation of Aircraft Maintenance Documentation in the International Aerospace Maintenance Language. BDC Techical Services, Slack Lane, Derby, 1985. AECMA Document PSC-85-16598.

[2] T. Briscoe and J. Carroll. Generalised Probabilistic LR Parsing of Natural Language (Corpora) with Unification-based Grammars. Technical report number 224, University of Cambridge Computer Laboratory, 1991.

[3] S. Caine and E. K. Gordon. PDL - a Tool for Software Design. In *AFIPS National Computer Conference*, volume 44, pages 271–76, Montvale, New Jersey, 1975. AFIPS Press of the American Federation of Information Processing Societies.

[4] J. Carroll, C. Grover, T. Briscoe, and B. Boguraev. A Development Environment for Large Natural Language Grammars. Technical report number 233, University of Cambridge Computer Laboratory, 1991.

[5] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[6] S. Douglas. Robust PATR for Error Detection and Correction. In A. Schöter and C. Vogel, editors, *Edinburgh Working Papers in Cognitive Science: Nonclassical Feature Systems*, volume 10, pages 139–155. unpublished, 1995.

[7] D. Duffy, C. MacNish, J. McDermid, and P. Morris. A framework for requirements analysis using automated reasoning. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *CAiSE*95: Proc. Seventh Advanced Conference on Information Systems Engineering*, Lecture Notes in Computer Science 932, pages 68–81. Springer-Verlag, 1995.

[8] J. Fain, J. G. Carbonell, P. J. Hayes, and S. N. Minton. MULTIPAR: A Robust Entity Oriented Parser. In *Proceedings of the 7th Annual Conference of The Cognitive Science Society*, 1985.

[9] P. J. Hayes. Flexible Parsing. *Computational Linguistics*, 7.4:232–241, 1981.

[10] J. E. Hoard, R. Wojclk, and K. Holzhauser. An Automated Grammar and Style Checker for Writers of Simplified English. In P. O. Holt and N. Williams, editors, *Computers and Writing: State of the Art*, pages 278–296. Kluwer Academic Publishers, Oxford, England, 1992.

[11] G. F. Luger and W. A. Stubblefield. *Artificial Intellegence: Structures and Strategies for Complex Problem Solving.* Benjamin Cummings, 1993.

[12] B. Macias and S. Pulman. *Natural language processing for requirements specifications*, chapter four, pages 67–89. Chapman and Hall, 1993.

[13] D. Magerman and C. Weir. Efficiency, Robustness and Accuracy in Picky Chart Parsing. In *Proceedings of the 30th ACL, University of Delaware, Newark, Delaware*, pages 40–47, 1992.

[14] M. Osborne. Parsing Computer Manuals using a Robust Alvey Natural Language Toolkit. International Workshop on Industrial Parsing of Software Manuals, Limerick, Ireland, May 1995.

[15] M. Osborne and D. Bridge. Learning unification-based grammars using the Spoken English Corpus. In *Grammatical Inference and Applications*, pages 260–270. Springer Verlag, 1994.

[16] S. Shubert, J. Spyridakis, M. Coney, and H. Holmback. Comprehensibility of Simplified English. In *STC Region 7 Conference Proceedings*, pages 73–75. 1994.

[17] C. Vogel and R. Cooper. Robust Chart Parsing with Mildly Inconsistent Feature Structures. In A. Schöter and C. Vogel, editors, *Edinburgh Working Papers in Cognitive Science: Nonclassical Feature Systems*, volume 10, pages 197–216. unpublished, 1995.

[18] R. M. Weischedel. Meta-rules as a Basis for Processing Ill-formed Input. *Computational Linguistics*, 9:161–177, 1983.

[19] V. S. Whitis and W. N. Chiang. A State Machine Development for Call-Processing. In *IEEE Electro '81 Conference*, pages 7/2–1–7/2–6, Washington D. C., 1981. Computer Society Press of the Institute of Electrical and Electronic Engineers.

[20] P. Zave and R. T. Yeh. Executable Requirements for Embedded Systems. In *Fifth IEEE International Conference on Software Engineering*, pages 295–304, Washington D. C, 1981. Computer Society Press of the Institute of Electrical and Electronic Engineers.