Natural Language Processing and Requirements Engineering: a Linguistics Perspective

By Dr. Christian R. Huyck (c.huyck@mdx.ac.uk)

&

Feroz Abbas

Abstract

This paper presents discusses some of the advantages that Natural Language Processing technology can bring to Requirements Engineering. This includes a discussion of ambiguity and underspecification. These problems are associated with Natural Language and can lead to misunderstandings in design specifications. Natural Language Processing techniques can easily detect these problems and suggest solutions.

This paper also presents a prototype system used for detecting ambiguity. This system has been evaluated and though it is clearly a prototype, it is capable of detecting ambiguity. This shows that one NLP tool for Requirements Engineering is viable.

Some sentences may be syntactically ambiguous, but all humans would view them as unambiguous. A mechanism for detecting syntactically ambiguous but semantically unambiguous sentences is suggested. This mechanism has the advantage of helping to develop a domain model of the document; this domain model can be used for Natural Language Processing, but can also be used as a primary component of the document specification.

Introduction

Natural Language Processing (NLP) has recently reached a stage of maturity where it is more and more industrially viable [Church 95]. Areas of research such as Machine Translate (MT), Speech Recognition, and Text Extraction are now in commercial applications. NLP is no longer merely a research task focused on simple examples. It now works with real people talking on the phone to machines, newspaper articles being scanned and manuals being corrected by machine.

Speech Recognition is a profound success. You can give instructions on the telephone to an answering machine by voice. Machine stenographers are available on a PC to translate speech into a letter. These systems function almost as well as humans, they are always available, and much less expensive.

Both MT and Text Extraction (TE) are commercially viable, however, a full-fledged translation of an important document is rarely left up to a machine. Instead, an expert translator passes the initial document through the MT system and then corrects the machine translation. This speeds the translation process by as much as 10 times. Text Extraction from Natural Language (NL) documents in a given domain functions at a high degree of precision and recall. Though this is below human functioning, it is quite close, and functions in a fraction of the time [MUC-6 1995, MUC-7 1998].

Can these successes be applied to Requirements Engineering (RE)? Others [Ryan 93] have noted that NLP will not solve all of RE’s problems. RE is more than a simple interpretation of NL text.

Requirements Acquisition is a complex process. This process involves a large amount of communication involving NL. NL is both ambiguous and underspecified. Simple NLP tools may be able to aid in communication between the Requirements Engineer and the Domain Expert, and aid in developing and maintaining appropriate RE documents.

In this paper we present the prototype of an NLP tool for ambiguity detection. This functions by using standard NLP parsing techniques to flag ambiguous sentences. This shows that NLP techniques can be used to aid RE.

Furthermore, NLP systems interact in sophisticated ways with the Domain Model. This Domain Model is both a key product of the RE process, and a key component in the process. Additionally, this Domain Model is a key component to the NLP system. Everything seems to depend on the Domain Model. Ideally, the Domain Model will be built by the Requirements Engineer with help from the NLP system.

Requirements Acquisition

The Requirements Acquisition task is an iterative process of discovery, refinement, and modelling leading to the creation of an artefact, the specification. This can on occasions be the subject of a contract between the system supplying organisation and the end user (customer)[Pressman 1997]. Typically the task involves at least 2 parties, a systems professional (Requirements Engineer) and a systems user (Domain Expert).

The process of interaction between these two parties is an information intensive activity and will involve the use of both spoken and written language. At its simplest a transcript (possibly verbatim) of an interview is converted into a written document and this document is then subjected to stepwise refinement, again possibly through further dialogue. The specification will eventually be written in a natural language assisted by some formal or semi-formal "artificial language".

Requirements Acquisition can lead to a document or documents that outline the requirements. However, a large amount of the information may not be written down, and is merely inside the Requirement Engineer’s head.

Communication involves one party trying to transmit his internal model to another party. The transmission does not necessarily (and rarely if ever does) contain the complete model. To the extent the internal model is specified, the process of communication has succeeded.

In the case of RE, more than simple one-way communication is needed. The Requirements Engineer often works with the domain expert to develop the planned system. The Requirements Engineer is actually participating in developing the model because the Domain Expert may not have knowledge of implementation details. Communication still takes place, but it is important that many unspoken assumptions are written down so that both speakers have a similar internal model of the problem and the proposed solution.

An NLP system is not going to replace the Requirements Engineer. However, it is possible that NLP systems can act as tools for him. They can translate NL into and from formal languages. NLP systems can help maintain the documents, and aid the expert in communicating with the users. These systems can speed the RE process, and help to find problems with the specification.

Ambiguity and Underspecification

The continued use of natural language to specify requirements is often accompanied by warnings of the inherent ambiguity of natural language. However, it is possible the problem will have arisen because the act of converting spoken discourse into its written counterpart could result in loss of information content. Another possibility is that this act created incorrect information.

Ambiguity is a problem that is difficult for NLP systems to handle. Given a sentence that has multiple interpretations, how do you select the correct interpretation? Humans, while processing NL, often overlook ambiguities. The correct interpretation is obvious to humans because they have access to information, such as semantics, that is not typically used in NLP systems.

When all readers derive the same interpretation, ambiguity is not a problem; when different readers derive different interpretations, the text can lead to problems, particularly in RE documents. For example, a sentence has two interpretations, *X* and *Y*. The Domain Expert means interpretation *X*, but the Requirements Engineer reads the sentence as interpretation *Y*. There may be a major problem due to this miscommunication. While ambiguity is a problem for the NLP interpretation of text, it is also a strength because NLP systems can easily find ambiguities. These ambiguities can be pointed out to the writer and can be corrected. This eliminates confusion in documents.

An underlying assumption in many conversations is that the participants are co-operating with each other. This principle was first set out by Grice [Grice 1975]. It is by no means obvious that this principle applies to all exchanges between Requirements Engineers and the Domain Experts, particularly in situations where the prospect of new systems is unwelcome. The existence of assumptions provides a further opportunity for misunderstandings to arise.

In other words, the way we communicate assumes a vast amount of ‘shared’ knowledge of how the world is. This poses problems when we attempt to use computers for NLP. This is underspecification in NL. This underspecification can also lead to problems when the ‘obvious’ interpretation differs between the Requirements Engineer and the Domain Expert. Again, this weakness in NLP systems can be turned into a strength, because NLP systems can easily find examples of underspecification and point them out to the Requirements Engineer.

Tools for Interacting with Requirements Documents and Domain Experts

While designing an Information System, many documents may be created. These documents are often interdependent. Moreover, these documents change over time leading to the problem of conflicting documents. Document dependencies along with NLP techniques can aid in maintaining these documents; NLP lexical techniques can be used to explain jargon; and NLP parsing techniques can be used to show ambiguities in design specifications.

One of the most common and dangerous problems in documents is that two people give different interpretations to the same string of words. This different understanding can lead to long-term confusion in the project. Technical writers are trained to avoid these ambiguities, but even the best text can be ambiguous. Ambiguity is a common problem for NLP. Humans tend to unconsciously remove a great deal of ambiguity while interpreting a sentence, but it is difficult for machine parsers to remove these ambiguities. However, it is easy for NL parsers to flag ambiguities. Once flagged, the writer can easily remove the ambiguities leading to a less ambiguous document.

An Ambiguity Flagging System

As a test of our ideas, we developed a simple system to detect and flag ambiguity in Requirements Specifications. This functioned by parsing the Requirements Specification, and flagging any sentences which had multiple syntactic interpretations.

When considering ambiguity, one thinks of syntax and semantics. Where syntax is concerned with the grammatical arrangement of words in a sentence, semantics deals with meanings of words and sentences. The prototype handled syntactic ambiguity.

The system was based on the LINK parser [Lytinen 92]. LINK uses a chart parser [Allen 87]. The input to the LINK system is a grammar, a lexicon, and sentences. LINK produces a chart describing all legal grammar rule applications over the given sentence.

The system was tested on randomly chosen sentences from a Requirements Specification written in natural language. Any sentences would have worked, but it is best to test sentences that are from the desired format. The Specification consisted of 10 sentences derived from the Flight Crew Operating Manuals of the A320 airbus. These sentences were used in [Ladkin 95].

The program makes use of complex set of unification-based grammar rules [Shieber 86]. The grammar was a modified version of the grammar used for [Huyck 98]. It was transformed from the initial format to one more suitable for chart parsing. During testing of the Specification sentences, the grammar was modified to increase its coverage. The final grammar consisted of 53 grammar rules. While this grammar was not a complete description of English, it did have substantial coverage.

In order for the grammar rules, and hence the system, to work, a lexicon was required. The particular one utilised was a relatively large lexicon, which formed the foundation of the system, and upon which it was dependent.

The charts corresponding to the sentences contained every plausible node combination (combinations validated by the grammar) in a sentence, as well as the total number of arguments and constituents for certain parts of the sentence. This included the total number of possibilities including the first and last nodes, that is, the entire sentence including the full stop. If a complete sentence had more than one possibility, then it was considered ambiguous.

We wanted the system to display sentences that were ambiguous, but ignore ones that were not. A sentence was ambiguous if it had more than one complete interpretation. It was not ambiguous if there were one or zero interpretations[1].

Each word has one or more senses, as it may be utilised in different situations. For a Requirements Specification document, the total number of senses may be very large.

[1] Some sentences had zero interpretations because the grammar would not necessarily give an interpretation for each sentence.

The word ‘saw’ was included in the lexicon, and is an interesting example, as it had 3 possible meanings. The first was a past tense of see, the second was the object saw and the third was the use of that object. This yielded 1 noun and 2 verbs, hence illustrating lexical ambiguity, not just by its multiple occurrence, but also because 2 of the occurrences were verbs.

Results

We were expecting each sentence to have at least one interpretation. However, when the program was executed the system perceived only 2 of the 10 sentences to be ambiguous. These were:

- ‘A hydraulics failure occurs if both the green and yellow hydraulic pressures are insufficient.’

- ‘Hydraulics are normal if both the green and yellow hydraulic pressures are OK.’

According to the particular grammar rules utilised, the first had 2 interpretations and the second had 8. The other 8 sentences had no complete interpretations.

The reason that the results were not as expected, and perhaps unusual, is due to the incomplete coverage of the grammar rules. The grammar had many rules but most did not apply to the 10 sentences. As a result, there were not enough possible combinations to generate complete interpretations of the sentences that we tested. This clearly suggests that in order to achieve accurate results, more time needs to be invested into the development of grammar rules.

As one can see, ambiguity is a real aspect of text-based Natural Language Processing. It is easy to speculate that with a more comprehensive grammar, the results would have been more accurate. However, by introducing a larger grammar, the amount of prospective ambiguity will also increase, thus creating even more problems.

Semantics Improves the System

An improvement on the prototype is to flag only sentences that are ‘truly’ ambiguous. That is, to flag sentences which different people might interpret differently.

Some sentences are syntactically ambiguous, but virtually every person would interpret them unambiguously [Ford et. al. 1982]. For example, Ford et. al. presented 20 subjects with syntactically ambiguous sentences. One sentence was:

The women discussed the dogs with the policemen.

Example 1.

All 20 subjects gave one interpretation to the sentence*: the women discussed with the policemen*. The other interpretation has the women discussing dogs that were with the policemen.

A tool that can flag ambiguous sentences is useful; however, a tool that can flag only truly ambiguous sentences is much more useful. It would reduce the need for the user of the tool to ignore many suggested changes. That is, a tool that flagged only sentences to which different people might give different interpretations would be better than a tool that flagged all ambiguous sentences.

Such a tool could take advantage of the semantic cohesion that people use while parsing sentences. For example, in Example 1, policemen are good participants in a discussion. They are much less good at being things that accompany dogs. This is not to say that they cannot accompany dogs, but they are much better at discussing as far as the participants of the study were concerned. Ford et al. presented many sentences all of which have two semantically plausible interpretations. Example 2 is syntactically ambiguous sentence that has only one semantically plausible interpretation.

I saw the girl with the boy.

Example 2.

In this example the boy is accompanying the girl. Another interpretation is that the boy was used to see the girl as in the sentence "I saw the girl with the telescope". This second interpretation is not very semantically plausible[2].

This tool would need to have a sophisticated semantics knowledge base. While much of this knowledge could be domain independent (eg. humans are good actors), much of it would be domain dependent. Consequently, developing this semantics knowledge base would aid in the development of the general domain knowledge base used in the RE process. This domain description could be a key component of the Requirements Specification. It is a formal description of the domain.

Conclusion

Ambiguity and underspecification are two key problems in Requirements Engineering documents. An ambiguous piece of text may be

[2] Other interpretations are plausible when the lexical ambiguity of “saw” is considered. “Saw” could be the action of cutting using a saw as opposed to the action of seeing. See section prototype undone

interpreted in one manner by the Requirements Engineer, and in another manner by the Domain Expert. This misunderstanding can lead to real problems. Similarly, NL text leaves things unsaid; it is underspecified. It is up to the reader to fill in the assumption. When the Requirements Engineer and the Domain Expert fill in the assumptions differently, there can be real problems.

NLP systems may help to solve this problem. They can flag ambiguous texts, and note where underspecification occurs. The Requirements Engineer can then either remove the ambiguity or underspecification, or at least agree with the Domain Expert as to the correct solution between ambiguities, or the missing information in underspecified text. Our prototype shows that this is technically viable, though more work is needed to make it an industrially viable program.

Recent advances in NLP have shown that NLP is at a state where it can be used in real world applications. While NLP systems are not the "silver bullet" that will solve the RE problem, NLP can usefully be used as an aid to RE.

Simple NLP tools can easily be used by Requirements Engineers to simplify their work and to act as simple checks. The prototype described in this paper can be either a stand alone system or it can be use as part of a suite of NL and RE tools. Obviously this would involve further work, but the prototype is an existence proof.

More advanced NLP tools could be used to help solidify the Domain Model, and to act as translators between various RE documents and formal models. These "more advance NLP tools" are currently just ideas. We have not implemented a system to automatically acquire Domain Knowledge from a Requirements Specification. However, we should be able to develop such a tool, and other tools. These tools will increase the efficiency and effectiveness of Requirements Engineers.

Bibliography

Abeysinghe, G. and Huyck, C. 1999. Process Modelling with Natural Language Input. International Conference on Enterpise Information Systems. Setubal, Portugal.

Advance Research Projects Agency. 1995. Proceedings of the Sixth Message Understanding Conference, Columbia, MD. August 1995. San Mateo, CA: Morgan Kaufmann Publishers.

Advance Research Projects Agency. 1998. Proceedings of the Seventh Message Understanding Conference, Fairfax, VA. May 1998. San Mateo, CA: Morgan Kaufmann Publishers.

Allen, James. 1987. Natural Language Understanding. The Benjamin/Cummings Publishing Company, Inc. Menlo Park, CA.

Chomsky, Noam. 1966. Syntactic Structures. Mouton and Co. The Haque

Church, Kenneth W. and Lisa F. Rau. 1995. Commercial Application of Natural Language Processing. In Communication of ACM 38:11 pp. 71-80.

Ford, Marilyn, Joan Bresnan and Ronald Kaplan. 1982. A competence-based theory of syntactic closure. In *The mental representation of grammatical relations,* ed. Joan Bresnan. Cambridge, MA: MIT Press

Grice, H.P. 1975. Logic and Conversation. In Syntax and Semantics 3: Speech Acts. Cole, P. and Morgan, J.P. (eds.) Academic Press

Huyck, Christian. 1998. The MUC7 Plink System. In Proceedings of the Seventh Message Understanding Conference, Fairfax, VA. May 1998. San Mateo, CA: Morgan Kaufmann Publishers.

Ladkin, P. B. Analysis of a Technical Description of the Airbus A320 Braking System, High Integrity Systems, 1(4):331-349, 1995.

Lytinen, Steven. 1992. A unification-based, integrated natural language processing system. Computers and mathematics with Applications, 23 (6-9), pp. 403-418.

Pressman, Roger. 1997. Software Engineering: A Practitioners Guide Fourth Edition.

Ryan, Kevin. 1993 The Role of Natural Language in Requirements Engineering. Proceeding of the IEEE International Symposium on Requirements Engineering. IEEE Computer Society Press.

Shieber, Stuart. 1986. An Introduction to Unification-Based Approaches to Grammar. CSLI Stanford, CA.

Sommerville, Ian. 1996. Software Engineering. Addison-Wesley. ISBN 0-201-42765-6

Yule, George. 1996. The Study of Language. Cambridge University Press ISBN 0521 56851