

# Contextual Knowledge Representation for Requirements Documents in Natural Language

Beum-Seuk Lee      Barrett R. Bryant

Department of Computer and Information Sciences  
The University of Alabama at Birmingham  
Birmingham, Alabama, U.S.A. 35294-1170  
{leebs, bryant}@cis.uab.edu

## Abstract

In software requirements engineering there have been very few attempts to automate the translation from a requirements document written in a natural language (NL) to one of the formal specification languages. One of the major reasons for this challenge comes from the ambiguity of the NL requirements documentation because NL depends heavily on context. To make a smooth transition from NL requirements to one of the formal specification languages we need a precise yet expressive knowledge representation that captures not only syntactic but also contextual information of the requirements. We propose the Contextual Natural Language Processing to overcome the ambiguity in NL using this contextual knowledge representation and Two-Level Grammar (TLG) to construct a bridge between a NL requirements specification and a formal specification to promote rapid prototyping and reusability of requirements documents.

## Problem Statement and Prior Research

When a complex system with heavy interactions among its components is to be built, first the requirements of the system are spelled out according to the desires of the stakeholders. Several formal specification languages have been developed to formally describe the system (Alagar & Periyasamy 1998) based on decomposition and abstraction information of the requirements. However still the natural language (NL) has remained as the practical choice for the domain experts to specify the system because those formal specification languages are not easy to master. In addition, the process of the elicitation and negotiation of the requirements is carried out usually in natural language. Therefore the requirements documentation written in NL has to be reinterpreted into a formal specification language by software engineers. Pohl rightly stated regarding this process that improving an opaque system comprehension into a complete system specification and transforming informal knowledge into formal representations are the major tasks in requirements engineering (Pohl 1993). When the system is very complicated, which is mostly

the case when one chooses to use formal specification, this conversion is both non-trivial and error-prone, if not implausible. The challenge of formalizing the requirements document results from many factors such as miscommunication between domain experts and engineers. However the major bottleneck of this conversion is from the inborn characteristic of ambiguity of NL and the different level of the formalism between the two domains of NL and the formal specification. This is why there have been very few attempts to automate the conversion from requirements documentation to a formal specification language.

To handle the problem of ambiguity and different formalisms, some have argued that the requirements document has to be written in a particular way to reduce ambiguity in the document (Wilson 1999). Others have proposed controlled natural languages (e.g., Attempto Controlled English (ACE) (Fuchs & Schwitter 1996)) which limit the syntax and semantics of NL to avoid the ambiguity problem. Even though the former approach provides a better documentation to work with, it hasn't accomplished any automated conversion from a natural language requirements document to a formal specification language. The latter has similar goals as ours to realize the automated conversion but restrictions on the syntax and semantics of the language result in losing the flexibility of NL. Also the user still has to remember the restrictions. Moreover the target language of this controlled language is PROLOG which is good for prototyping but lacks important properties such as strong typing to be used as a formal specification language. Another approach to natural language requirements analysis is to search each line of the requirements document for specific words and phrases for the purpose of quality analysis (Wilson, Rosenberg, & Hyatt 1996). A similar project (Girardi 1996) focuses mainly on the automatic indexing and reuse of the software components in the requirements documents.

In summary, the linguistic descriptions in the requirements document as the inputs for these systems have been too restricted and controlled. Also the related research so far, only focusing on the validation and verification of requirements, has not achieved a full conversion of the requirements specifications into a formal

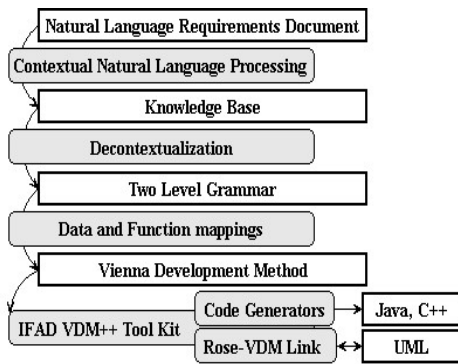


Figure 1: System Structure.

specification and implementation of the specifications.

## Project Approach

In our research project, Contextual Natural Language Processing (CNLP) (McCarthy 1993) is used to handle the ambiguity problem in NL and Two Level Grammar (TLG) (van Wijngaarden 1965) is used to deal with the different formalism level between NL and formal specification languages to achieve the automated conversion from NL requirements documentation into a formal specification (in our case VDM++, an object-oriented extension of the Vienna Development Method (Dürr & van Katwijk 1992)).

First a knowledge base is built from the requirements documentation in NL using the contextual natural language processing by parsing the documentation and storing the syntax, semantics, and pragmatics information. In this phase, the ambiguity is detected and resolved, if possible. Then the knowledge base is converted into TLG by removing the contextual dependency in the knowledge base. Finally the TLG code is translated into VDM++ by data and function mappings. Once VDM++ representation of the specification is acquired we can convert this into programming languages such as Java<sup>TM</sup> or C++, using the code generator that the VDM Toolkit<sup>TM</sup> provides and into a model in the Unified Modeling Language (UML) (Quatrani 2000) using a link with Rational Rose<sup>TM</sup> (IFAD 2000). The entire system structure is shown in Figure 1.

This paper concentrates on how the knowledge base is built from a requirements document and on how the knowledge base is converted into TLG. The translation of TLG to VDM++ has been fully developed and described in detail in another publication.

The following simple specification of an Automatic Teller Machine (ATM) will be used as the running example throughout the paper to illustrate the system.

Bank keeps list of accounts. It verifies ID and PIN giving the balance and updates the balance with ID. An account has three data fields; ID, PIN, and balance. ID and PIN are integers and

balance is a real number.

ATM has 3 service types; withdraw, deposit, and balance check. For each service first it verifies ID and PIN from the bank giving the balance. ATM withdraws an amount with ID and PIN giving the balance in the following sequence. If the amount is less than or equal to the balance then it decreases the balance by the amount. And then it updates the balance in the bank with ID. ATM deposits an amount with ID and PIN giving the balance in the following order. It increases the balance by amount and then updates the balance in the bank with ID. ATM checks the balance with ID and PIN giving the balance.

## Construction of Knowledge Base from Requirements Document

The raw information of the requirements document in natural language is not proper to be used directly because of the ambiguity and implicit semantics in the document. Therefore an explicit and declarative representation (knowledge base) is needed to represent, maintain, and manipulate knowledge about a system domain (Lakemeyer & Nebel 1994). Not only does the knowledge base have to be expressive enough to capture all the critical information but also it has to be precise enough to clarify the meaning of each knowledge entity (sentence). In addition, the knowledge base has to reflect the structure of TLG into which the knowledge base is translated later.

It is worthwhile to mention that our approach, different from most of the other commonsense reasoning approaches ((Borgida & Etherington 1989), (Baral & Gelfond 1994), and (Gogic *et al.* 1995) just to name a few), defers the reasoning (inference) process of knowledge base until the knowledge base is converted to a more formal representation even though contradiction is handled at the time of knowledge base construction. This logical (and even procedural) process is carried out in the form of prototyping of specifications in VDM++.

The knowledge base isn't a simple list of sentences in the requirements document. The linguistic information of each sentence such as lexical, syntactic, semantic, and most importantly discourse level information has to be stored with proper systematic structure.

Each sentence in the requirements document is read by the system and tokenized into words. At the syntactical level, the part of speech (e.g. noun, verb, adjective) of each word is determined by bottom-up parsing, whereas the part of sentence (e.g. subject and object) of each word is determined by top-down parsing (Jurafsky & Martin 2000). Our parsing algorithm is based on this approach, with slight modification for better accuracy and control. Separating the parsing process into these two different sub-processes makes the algorithm simpler because the latter process is very context-sensitive about the features like verb form and sub-categorization whereas the former one is context-sensitive about person and number features (Gazdar *et al.* 1985). By using the predetermined part of speech for each word from the

part-of-speech parsing, the number of the rules for the context free grammar for the part-of-sentence parsing is reduced substantially. The corpora of statistically ordered parts of speech (frequently used ones being listed first) of about 85000 words from Moby Part-of-Speech II (Grady 1994) are used to resolve the syntactic ambiguity when there is more than one valid parsing tree. Also elliptical compound phrases, comparative phrases, compound nouns, and relative phrases are handled in this phase as well.

Also the anaphoric references (pronouns) in a sentence are identified according to the current context history. A pronoun can represent a word, sentence, or even context. This is done according to the recency constraints (the recent word has a higher priority than less recent ones) and the discourse focus (the co-referred one has a higher priority than ones that aren't) (Brennan, Friedman, & Pollard 1987) (Grosz, Joshi, & Weinstein 1983).

Once the references of pronouns are determined, each sentence is stored into the proper context in the knowledge base. This involves the syntactic, semantic, and most importantly discourse level information. This part of the project is the most challenging part because if a sentence is located in a long context, the meaning of the sentence can totally change than what is originally intended. A contextual knowledge base is formalized as a tree-like data structure not only to store each sentence in its right context but also to make a smooth conversion from the knowledge base to TLG. Meta-level context (context for context) determines where to put each sentence in the tree according to the discourse level information.

The current context is created or switched dynamically according to the discourse level information (sections, subsections, and paragraphs) and semantics information in related sentences. For instance, in the ATM example the phrase “in the following sequence” indicates that the following sentences are likely to stay within the current context. Therefore a sub-context to hold the following sentences has to be created under the current context. Each context keeps a list of keywords. For a sentence to belong to a context, at least one significant word in the sentence has to be an element of the keywords list of the context. This is similar to the frame problem (McCarthy & Hayes 1987) in the sense that given a current situation (context) and a new action (sentence) a new situation (context) is to be identified. Contradictions are resolved by not allowing two contradictory sentences under the same context.

The contextual structure of the knowledge base is shown in the Figure 2. The black ovals indicate the contexts that hold the data type information whereas the gray ovals indicate the contexts that contain the functional information. Note how the meaning of the sentence “It increases the balance by amount” can be clarified further by referring to its outer contexts. Therefore we can tell from Figure 2 that this decrement operation is a part of the deposit service and this service in turn

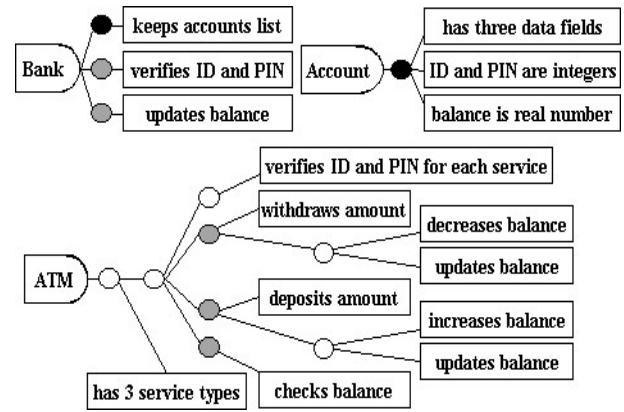


Figure 2: Knowledge base for ATM.

belongs to ATM.

In our research, a lexical database, WordNet (Miller 1990), is used in several places. To resolve anaphoric references, categories of nouns (event, attribute, act, object, location, etc.) and verbs (motion, possession, stative, etc.) are used for a better judgement. For example, in the sentence “a dog eats a cookie and it likes it” the word ‘dog’ is a noun in animal category, the word ‘cookie’ is a noun in food category, and the word ‘eats’ is a verb in consumption category. Therefore the first ‘it’ refers to ‘dog’ which consumes the second ‘it’ which refers to ‘cookie’. Also hypernym information for nouns and verbs from WordNet are used for keyword checking in the context construction and for similarity checking of many-to-one mappings in the TLG translation which will be discussed shortly in the next section. As an example the word ‘computers’ is closer to the word ‘machines’ than the word ‘banks’ in the sentence “computers are used in banks and the machines are efficient.”

In summary, a contextual knowledge representation is constructed from a requirements document capturing not only syntactic and semantic information but also structured contextual information. Along with this process, linguistic ambiguity is detected and resolved in parsing and construction of the contextual knowledge base.

### Translation from Knowledge Base into Two-Level Grammar

Two-Level Grammar (TLG) may be used to achieve translation from an informal NL specification into a formal specification. Even though TLG has NL-like syntax its notation is formal enough to allow formal specifications to be constructed using the notation. It is able not only to capture the abstraction of the requirements but also to preserve the detailed information for implementation. The term “two level” comes from the fact that a set of domains may be defined using context-free grammar, which may then be used as arguments in predicate functions defined using another grammar. The combi-

nation of these two levels of grammar produces Turing equivalence (Sintzoff 1967) and so TLG may be used to model any type of software specification. The basic functional/logic programming model of TLG is extended to include object-oriented programming features suitable for modern software specification. The syntax of the object-oriented TLG is:

```
class Class_Name.
  Data_Name {,Data_Name}::Data_Type {,Data_Type}.
  Rule_Name : Rule_Body {, Rule_Body}.
end class [Class_Name].
```

where the term that is enclosed in the curly brackets is optional and can be repeated many times, as in Extended Backus-Naur Form (EBNF). The data types of TLG are fairly standard, including both scalar and structured types, as well as types defined by other class definitions. The rules are expressed in NL with the data types used as variables.

Once the knowledge base is built from the requirements document the knowledge base is converted into TLG. This conversion from the knowledge base to TLG flows very nicely because the knowledge base is built with the structure taking this translation into consideration. The tree-like contextual structure of the knowledge base conveys the abstracted representation of the requirements document and this abstraction makes the conversion straightforward.

First identifying each class and then collecting its data members and functions into the class from the knowledge base is the major task of the conversion. The root of each context tree becomes a class. Then the body of each class is built up with the information in the sub-contexts of each root. The knowledge base of the ATM specification would be translated into the following ATM class in TLG.

```
class ATM.
  Balance :: Float.
  Amount :: Float.
  ID :: Integer.
  PIN :: Integer.

  withdraw Amount with ID and PIN giving Balance :
    verify ID and PIN from Bank giving Balance,
    if Amount <= Balance then
      Balance := ( Balance - Amount ),
      update Balance in Bank with ID
    endif.

  deposit Amount with ID and PIN giving Balance :
    verify ID and PIN from Bank giving Balance,
    Balance := ( Balance + Amount ),
    update Balance in Bank with ID.

  check balance with ID and PIN giving Balance :
    verify ID and PIN from Bank giving Balance.
end class.
```

Given a context with its sentences, first the types of the sentences are determined. A sentence can be a data type declaration, a rule, a statement for a rule, or a meta sentence which contains information about the

class itself or a set of rules. For example the sentence “checks balance” is a rule, the sentence “verifies ID and PIN for each service” is a meta information (for each service rule in this class, the verification statement has to be called first), and the sentence “increases balance” is a statement of the rule “deposits amount” in Figure 2. In addition, a statement for a rule can be a function call statement, a logic statement, or language specific statement (if-then-else statement, assign statement, and etc.). In the withdraw rule, there are an assign statement and a function call statement in an if-then statement. Usually the main verb of a sentence determines the type of the sentence whereas the objects of the sentence is used as parameters. Not only the linguistic information (parts of sentence, hypernyms, and so forth) but also the contextual structure of the knowledge base makes this classification of each sentence possible.

When the system proceeds with the ATM knowledge base, it detects the fact that the data type of ‘Amount’ hasn’t been specified in any place of the requirements document. So it asks for its manual input from the user. Also observe that the sentence that increases or decreases the balance is mapped into the TLG assign statement. NL has a fairly large size of vocabularies whereas TLG uses specific words for the language-defined operations. Therefore there is a many-to-one mapping between a NL expression and a specific TLG operation just like the assign operation example.

Once we have translated the knowledge base into TLG and then the TLG specification into a VDM++ specification (we refer the readers to (Bryant & Lee 2002) for more details on this translation) we can convert this into a high level language such as Java<sup>TM</sup> or C++, using the code generator that the VDM Toolkit<sup>TM</sup> provides. Not only is this code quite efficient, but it may be executed, thereby allowing a proxy execution of the requirements. This allows for a rapid prototyping of the original requirements so that these may be refined further in future iterations. Another advantage of this approach is that the VDM Toolkit also provides for a translation into a model in the Unified Modeling Language (UML) using a link with Rational Rose<sup>TM</sup>.

## Conclusion

This research project is developed as an application of formal specification and linguistic techniques to automate the conversion from a requirements document written in NL to a formal specification language. The knowledge base is built up from a NL requirements document in order to capture the contextual information from the document while handling the ambiguity problem and to optimize the process of its translation into a TLG specification. Well structured and formalized data representations especially for the context are used to make smooth translations from NL requirements into the knowledge base and then from the knowledge base into a TLG specification. Due to its NL-like syntax and flexibility without losing its formalism, TLG is chosen

as a formal specification to fill the gap between the different level of formalisms of NL and formal specification language. With this approach we can achieve an executable NL specification for a rapid prototyping and reusability of requirements, as well as development of a final implementation.

**Acknowledgements.** This material is based upon work supported by, or in part by, the U. S. Army Research Laboratory and the U. S. Army Research Office under contract/grant number DAAD19-00-1-0350 and by the U. S. Office of Naval Research under award number N00014-01-1-0746. The authors would like to thank IFAD for providing an academic license to the IFAD VDM Toolbox in order to conduct this research.

## References

- Alagar, V. S., and Periyasamy, K. 1998. *Specification of Software Systems*. Springer-Verlag.
- Baral, C., and Gelfond, M. 1994. Logic programming and knowledge representation. *The Journal of Logic Programming* 19 & 20:73–148.
- Borgida, A., and Etherington, D. W. 1989. Hierarchical knowledge bases and efficient disjunctive reasoning. In Brachman, R. J.; Levesque, H. J.; and Reiter, R., eds., *KR'89: Principles of Knowledge Representation and Reasoning*. San Mateo, California: Morgan Kaufmann. 33–43.
- Brennan, S.; Friedman, L.; and Pollard, C. 1987. A Centering Approach to Pronouns. *Proc. 25th ACL Annual Meeting* 155–162.
- Bryant, B. R., and Lee, B.-S. 2002. Two-Level Grammar as an Object-Oriented Requirements Specification Language. *Proc. 35th Hawaii Int. Conf. System Sciences*.
- Dürr, E. H., and van Katwijk, J. 1992. VDM++ - A Formal Specification Language for Object-Oriented Designs. *Proc. TOOLS USA '92, 1992 Technology of Object-Oriented Languages and Systems USA Conf.* 63–278.
- Fuchs, N. E., and Schwitter, R. 1996. Attempto Controlled English (ACE). *Proc. CLAW 96, 1st Int. Workshop Controlled Language Applications*.
- Gazdar, G.; Klein, E.; Pullum, G. K.; and Sag, I. 1985. *Generalized Phrase Structure Grammar*. Basil Blackwell.
- Girardi, M. R. 1996. *Classification and Retrieval of Software through their Description in Natural Language*. Ph.D. Dissertation, Computer Science Department University of Geneva, Switzerland.
- Gogic, G.; Kautz, H. A.; Papadimitriou, C.; and Selman, B. 1995. The comparative linguistics of knowledge representation. In Mellish, C., ed., *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 862–869. San Francisco: Morgan Kaufmann.
- Grady, W. 1994. Moby Part-of-Speech II (data file).
- Grosz, B. J.; Joshi, A. K.; and Weinstein, S. 1983. Providing a Unified Account of Definite Noun Phrases in Discourse. *Proc. 21st ACL Annual Meeting* 155–162:44–50.
- IFAD. 2000. The VDM++ Toolbox User Manual. Technical report, IFAD (<http://www.ifad.dk>).
- Jurafsky, D., and Martin, J. 2000. *Speech and Language Processing*. Prentice Hall.
- Lakemeyer, G., and Nebel, B. 1994. *Foundations of knowledge representation and reasoning*, volume 810. Springer-Verlag Inc.
- McCarthy, J., and Hayes, P. J. 1987. *Some Philosophical Problems from the Standpoint of Artificial Intelligence*. Los Altos, CA: Kaufmann.
- McCarthy, J. 1993. Notes On Formalizing Context. Technical report, Computer Science Department, Stanford University, Stanford, CA.
- Miller, G. 1990. Wordnet: An On-line Lexical Database. *International Journal of Lexicography* 4(3).
- Pohl, K. 1993. The Three Dimensions of Requirements Engineering.
- Quatrani, T. 2000. *Visual Modeling with Rational Rose 2000 and UML*. Addison-Wesley.
- Sintzoff, M. 1967. Existence of van Wijngaarden's Syntax for Every Recursively Enumerable Set. *Ann. Soc. Sci. Bruxelles* 2 115–118.
- van Wijngaarden, A. 1965. *Orthogonal Design and Description of a Formal Language*. Mathematisch Centrum, Amsterdam.
- Wilson, W. M.; Rosenberg, L. H.; and Hyatt, L. E. 1996. Automated Quality Analysis Of Natural Language Requirement Specifications. Technical report, Naval Research Laboratory.
- Wilson, W. M. 1999. Writing Effective Natural Language Requirements Specifications. Technical report, Naval Research Laboratory.