

# Contextual Natural Language Processing and DAML for Understanding Software Requirements Specifications

Beum-Seuk Lee and Barrett R. Bryant

Department of Computer and Information Sciences

The University of Alabama at Birmingham

Birmingham, AL 35294-1170 U. S. A.

{leeb, bryant}@cis.uab.edu

## Abstract

In software engineering a system requirements document written in a natural language (NL) needs to be translated into one of the formal specification languages for system execution. When this translation is to be automated, resolution of the ambiguity in the document and explicit definition of implicit domain knowledge are necessary. In our approach, Contextual Natural Language Processing is used to overcome the ambiguity and the domain knowledge is expressed in DARPA Agent Markup Language (DAML). The result is a formal representation of the informal requirements in NL for prototyping and even for implementation.

## 1 Introduction

In software development still the natural language (NL) has remained as the practical choice for the domain experts to specify the system even with existence of many formal specification languages. This is due to the fact that formal specification languages are hard to master and inappropriate as a communication medium. However the syntax and semantics of NL, even with its flexibility and representation power, is not formal enough to be directly used for verification, prototyping, or implementation of the system. Therefore the requirements document in NL is translated, usually manually, into a formal specification. However, 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 major bottleneck of the automation of this conversion results from the inborn characteristic of ambiguity of NL and the implicit domain knowledge.

To handle this ambiguity problem, 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 and Schwitter, 1996)) which limit the syntax and semantics of NL to avoid the ambiguity problem. Another approach to NL requirements analysis is to search each line of the requirements document for specific words and phrases for the purpose of quality analysis (Wilson et al., 1996). A similar project (Girardi, 1996) focuses mainly on the automatic indexing and reuse of the software components in the requirement documents. However there has been no attempt to automate the conversion from requirements documentation into a formal specification language for implementation.

In our research project, Contextual Natural Language Processing (CNLP) is used to handle the ambiguity problem in NL and DARPA Agent Markup Language (DAML) (van Harmelen et al., 2000) is used to deal with the implicit domain knowledge 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 (Bjørner and Jones, 1978)).

First the requirements document is converted into Extensible Markup Language (XML) (Bray et al., 2000) format. Then a knowledge base is built from the XML requirements document using CNLP to parse the documentation and to store the syntax, semantics, and pragmatics information. In this phase, the ambiguity is detected and resolved, if possible. Once the knowledge base is constructed, its content can be queried in NL. The information of the domain specific knowledge specified in DAML is extracted. Next the knowledge base is con-

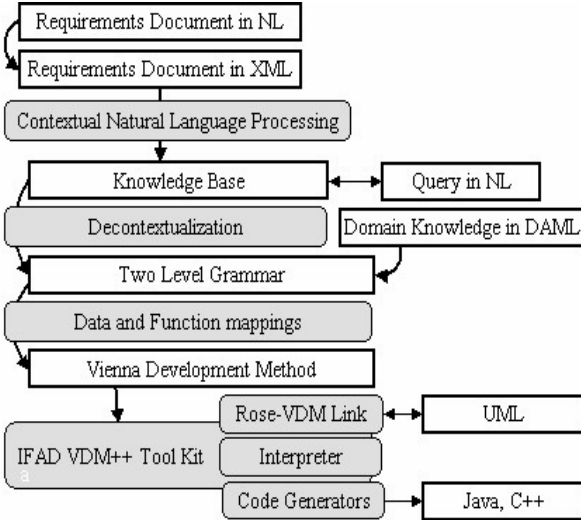


Figure 1: System Structure.

verted, with the information of the domain specific knowledge, into Two Level Grammar (TLG) by removing the contextual dependency in the knowledge base. TLG is used to build a bridge between the informal knowledge base and the formal VDM++ representation. Finally the TLG code is translated into VDM++ by data and function mappings. VDM++ has been our choice as a target specification language for this project because VDM++ has many similarities in structure to TLG and also has a good collection of tools for analysis and code generation. Once VDM++ representation of the specification is acquired we can do prototyping of the specification using the VDM++ interpreter. Also we can convert this into a high level language such as Java<sup>TM</sup> or C++ or into a model in the Unified Modeling Language (UML) (Quatrani, 2000) using the VDM++ Toolkit (IFAD, 2000). The entire system structure is shown in Figure 1.

In the sections which follow, we will present the following simple Automatic Teller Machine (ATM) example to illustrate our approach and describe the various system components.

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. The machine 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.

## 2 Construction of Knowledge Base from Requirements

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 and 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. 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.

To accomplish this, first 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 and Martin, 2000). 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 of 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 in this phase. In other words, when there is more than one valid parsing tree for a sentence, this corpora is used to break the tie. Elliptical compound phrases, comparative phrases, compound nouns, and relative phrases are handled in this phase as well. A part of the result of this process for the ATM example is shown as follows.

```
Bank keeps list of accounts
Part of speech : bank(noun) keeps(verb)
accounts_list(noun)
Part of sentence : ( subject verb object )

It verifies ID and PIN giving the balance and
updates the balance with ID
Part of speech : it(pronoun) verifies(verb) ID
and PIN(noun) giving(verb) the(article)
balance(noun)
Part of sentence : ( subject verb object
helping:( verb adjective object ) )
Part of speech : it(pronoun) updates(verb)
the(article) balance(noun) with(preposition)
ID(noun)
Part of sentence : ( subject verb adjective
object adverb preposition_object )
```

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 word has a higher priority than words that aren't) (Brennan et al., 1987) (Grosz et al., 1983). For a pronoun in a sub-sentence, first the nouns in the main sentence are checked. Also the previous subjects, objects, and objects for prepositions are checked in that order because there is stronger tendency for a pronoun to refer to the previous subject in a requirements document compared with other types of textual documents.

It is worthwhile to mention that the requirements documents are easier to process than other types of textual documents in the sense that usually requirements documents have well defined structures with less ambiguities and infrequent use of pronouns.

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 certain 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 and 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 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 belongs to ATM.

In our research, a lexical database, WordNet (Miller, 1990), is used in several places. To resolve anaphoric references (by distinguishing living things from others to resolve the pronouns

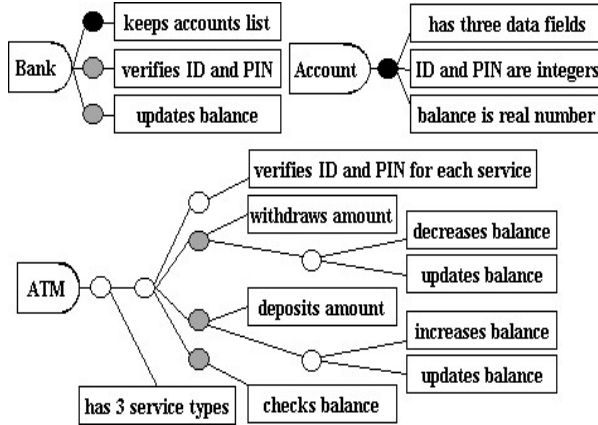


Figure 2: Knowledge base for ATM.

he and she), categories of nouns (event, attribute, act, object, location, etc.) and verbs (motion, possession, stative, etc.) are used. 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 keywords checking in the context construction. 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.” Another place where hypernym information for nouns from WordNet are used is the non-class words filtering in the TLG translation which will be discussed further shortly.

Also the content of the knowledge base can be queried by the user in natural language. The following dialogue between the system and a user shows some example queries about ATM.

User : What does the bank keep?  
 System : Bank keeps list of accounts.  
 User : How does the ATM deposit the amount?  
 System : ATM deposits amount with ID and PIN giving balance in following order, ATM increases balance by Amount, then ATM updates balance in bank with ID

Because the requirements are stored in a structural format according to the context, the relevant other information is also retrieved as shown in the answer for the second query.

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.

### 3 Domain Specific Knowledge in DAML

A requirements document usually contains specific information about how the system should work whereas the domain knowledge describes the relationship between components and other constraints which are usually presumed in requirements documents or too implicit to be extracted easily from the original documents. For example, the requirements says “the user inputs the 4 digit PIN number by pressing the buttons.” And the fact that the set of the buttons is a component of the ATM machine is implicitly assumed and therefore not explicitly mentioned in the requirements documents. So this kind of information is needed to be specified as the domain specific knowledge. The units of measurements, who passes what to whom, which synonyms of a word are used, what each acronym stands for, etc., are some of the examples of the domain specific knowledge that can supplement the requirements documents.

In our research the domain knowledge of a system is specified in DAML which is a frame-based language with an expressive semantics to facilitate the concept of the Semantic Web (Decker et al., 2000).

The following examples show the use of DAML as domain knowledge for the ATM example. The ‘disjointUnionOf’ notation in DAML can be used to list the subcomponents of a component. Three data fields of an account are shown as follow.

```
<daml:Class rdf:ID="Account">
  <daml:disjointUnionOf
    rdf:parseType="daml:collection">
    <daml:Class rdf:ID="ID"/>
    <daml:Class rdf:ID="PIN"/>
    <daml:Class rdf:ID="Balance"/>
  </daml:disjointUnionOf>
</daml:Class>
```

where rdf stands for Resource Description Framework on which DAML and XML are built.

The ‘sameClassAs’ definition in DAML can be used to indicate that the word ‘Machine’ used in the ATM requirement is a synonym of the word ‘ATM’ and that the word ‘ATM’ stands for “Automatic Teller Machine”.

```
<daml:Class rdf:ID="Automatic_Teller_Machine">
  <daml:sameClassAs rdf:ID="Machine"/>
  <daml:sameClassAs rdf:ID="ATM"/>
</daml:Class>
```

Using ‘ObjectProperty’ notation in DAML, the fact that Balance is passed from Bank to ATM can be expressed as follows :

```
<daml:ObjectProperty rdf:ID="passBalance">
  <rdfs:domain rdf:ID="Bank"/>
  <rdfs:range rdf:ID="ATM"/>
</daml:ObjectProperty>
```

The data type or the measurement unit of a component can be expressed using ‘Datatype-Property’ notation in DAML as shown below for the type of Amount.

```
<daml:DatatypeProperty rdf:ID="Amount">
  <rdfs:range rdf:resource="http://www.w3.org/
  2000/10/XMLSchema#float"/>
</daml:DatatypeProperty>
</daml:Class>
```

In summary the precise formal semantics of DAML provides a very useful way to specify the domain specific knowledge explicitly. This knowledge is used as supplementary information for the conversion from knowledge base to TLG.

#### 4 Conversion from Knowledge Base to TLG

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 combination 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 (Bryant, 2000).

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.

The 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 root of each context tree of the knowledge base becomes a class. And then the body of each class is built up with the sentence information in the sub-contexts of the root. The knowledge base of the ATM example in Figure 2 would be translated into the following TLG specification.

```
class Bank.
  Accounts_List :: AccountList.
  ID :: Integer.
  PIN :: Integer.
  Balance :: Float.

  verify ID and PIN giving Balance.
  update Balance with ID.
end class.

class Account.
  ID :: Integer.
  PIN :: Integer.
  Balance :: Float.
end class.

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.

```

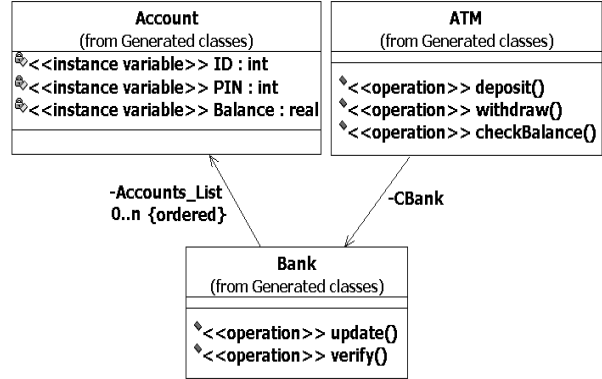


Figure 3: UML for ATM.

Observe that the sentence that increases or decreases the balance is mapped into the TLG assign statement. NL has a fairly large size vocabulary 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. Therefore this mapping function has to be defined before the translation takes place. As seen in the above TLG specification of ATM, TLG has flexibility with its NL-like syntax as well as formality with its strong typing and formal semantics.

Also not all the nouns can be candidates as class names. A word that is perceptive, cognitive, or stative (e.g. notation, fact) is likely to be in the non-class type whereas a word that represents artifact or substance (e.g. airplane, basket) is the class type. We use the hypernyms in WordNet to decide which type a noun belongs to and to filter out the non-class type words. Some of the sentences in the requirements documents are just commentarial and normally the subjects of these sentences can be categorized as the non-class type.

Once we have translated the knowledge base into TLG and then the TLG specification into a VDM++ specification (for more details on this translation we refer the readers to (Bryant and Lee, 2002)) 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. Namely the logical inconsis-

tencies, contradictions, and ambiguities hidden in the informal description can be discovered in the formal representation using the VDM++ Toolkit. 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> (Figure 3).

## 5 Summary and 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. Domain specific knowledge is represented in DAML to supplement this automation by specifying implicit domain knowledge explicitly. 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 to fill the gap between the different level of formalisms of NL and formal specification language.

The system can currently handle the presented example completely, as described. We are performing additional evaluations of the system for other requirements documents, including some requirements documents de-

scribing actual U. S. Army systems. It is expected that the technology we are developing will be applicable to these requirements documents as well. If successful, this will provide a very useful tool to assist software engineers in moving from the requirements document to the formal specification. Our future work is to continue developing the system to improve system usability and robustness with respect to its coverage of requirements documents. When finalized, it is expected that by using the formalized context in CNLP and TLG as a bridge between the requirements document and a formal specification language, we can achieve an executable NL specification for a rapid prototyping 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

- D. Bjørner and C. B. Jones. 1978. *The Vienna Development Method: The Meta-Language*. Springer-Verlag.
- T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. 2000. Extensible Markup Language (XML) 1.0 (Second Edition). Technical report, W3C (<http://www.w3c.org/xml>).
- S. Brennan, L. Friedman, and C. Pollard. 1987. A Centering Approach to Pronouns. *Proc. 25th ACL Annual Meeting*, pages 155–162.
- B. R. Bryant and B.-S. Lee. 2002. Two-Level Grammar as an Object-Oriented Requirements Specification Language. *Proc. 35th Hawaii Int. Conf. System Sciences*, Jan.
- B. R. Bryant. 2000. Object-Oriented Natural Language Requirements Specification. *Proc. ACSC 2000, 23rd Australasian Comp. Sci. Conf.*, pages 24–30.
- S. Decker, S. Melnik, F. van Harmelen, D. Fensel, M. C. A. Klein, J. Broekstra, M. Erdmann, and I. Horrocks. 2000. The semantic web: The roles of XML and RDF. *IEEE Internet Computing*, 4(5):63–74.
- N. E. Fuchs and R. Schwitter. 1996. Attempto Controlled English (ACE). *Proc. CLAW 96, 1st Int. Workshop Controlled Language Applications*.
- G. Gazdar, E. Klein, G. K. Pullum, and I. Sag. 1985. *Generalized Phrase Structure Grammar*. Brasil Blackwell.
- M. R. Girardi. 1996. *Classification and Retrieval of Software through their Description in Natural Language*. Ph.D. thesis, Computer Science Department University of Geneva, Switzerland.
- W. Grady. 1994. Moby Part-of-Speech II (data file).
- B. J. Grosz, A. K. Joshi, and S. Weinstein. 1983. Providing a Unified Account of Definite Noun Phrases in Discourse. *Proc. 21st ACL Annual Meeting*, pages 44–50.
- F. van Harmelen, P. Patel-Schneider, and I. Horrocks. 2000. Reference description of the DAML+OIL. Technical report, ([www.daml.org](http://www.daml.org)).
- IFAD. 2000. The VDM++ Toolbox User Manual. Technical report, IFAD (<http://www.ifad.dk>).
- D. Jurafsky and J. Martin. 2000. *Speech and Language Processing*. Prentice-Hall.
- G. Lakemeyer and B. Nebel. 1994. *Foundations of knowledge representation and reasoning*, volume 810. Springer-Verlag Inc.
- J. McCarthy and P. J. Hayes. 1987. *Some Philosophical Problems from the Standpoint of Artificial Intelligence*. Kaufmann, Los Altos, CA.
- G. Miller. 1990. Wordnet: An On-line Lexical Database. *International Journal of Lexicography*, 4(3).
- T. Quatrani. 2000. *Visual Modeling with Rational Rose 2000 and UML*. Addison-Wesley.
- M. Sintzoff. 1967. Existence of van Wijngaarden's Syntax for Every Recursively Enumerable Set. *Ann. Soc. Sci. Bruxelles 2*, pages 115–118.
- W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt. 1996. Automated Quality Analysis Of Natural Language Requirement Specifications. Technical report, Naval Research Laboratory.
- W. M. Wilson. 1999. Writing Effective Natural Language Requirements Specifications. Technical report, Naval Research Laboratory.