Automated Conversion from Requirements Documentation to an Object-Oriented Formal Specification Language [*]

Beum-Seuk Lee
Dept. Computer and Information Sciences
The University of Alabama at Birmingham
Birmingham, Alabama, U.S.A. 35294-1170
leebs@cis.uab.edu

Barrett R. Bryant
Dept. Computer and Information Sciences
The University of Alabama at Birmingham
Birmingham, Alabama, U.S.A. 35294-1170
bryant@cis.uab.edu

ABSTRACT
In software 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. We use Contextual Natural Language Processing (CNLP) to overcome the ambiguity in NL, and Two-Level Grammar (TLG) to construct a bridge between a NL requirements specification and a formal specification in VDM++, an object-oriented extension of the Vienna Development Method. The result is a system for mapping natural language requirements documents into an object-oriented formal specification language.

Keywords
Contextual Natural Language Processing, Object-Oriented Software Specification, Two-Level Grammar, Vienna Development Method, XML

1. INTRODUCTION

Recently many formal specification languages have been developed to handle their complex systems with heavy interactions between its components by decomposition and abstraction of the requirements of the system [1]. However still the natural language (NL) has remained as the practical choice for the domain experts to specify the system because formal specification languages are not easy to master. Even though NL is inherently object-oriented and descriptive with representation power, its semantics and syntax are not for-

mal enough to be used directly as a specification language. Therefore the requirements documentation usually written in NL has to be reinterpreted by software engineers into a formal specification language. 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. This challenge comes 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 this ambiguity problem, some have argued that the requirements document has to be written in a particular way to reduce ambiguity in the document [20]. Others have proposed controlled natural languages [10] 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 not a formal specification language but PROLOG which is good for prototyping but not for implementation.

In our research project, Contextual Natural Language Processing (CNLP) [17] is used to handle the ambiguity problem in NL and Two Level Grammar (TLG) [19] 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 the Vienna Development Method - VDM [3]). First the requirements document is converted into Extensible Markup Language (XML) [4] format. Then a knowledge base is built from this XML requirements documentation 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. Once the knowledge base is constructed, its content can be displayed in XML or queried in NL. Next the knowledge base is con-
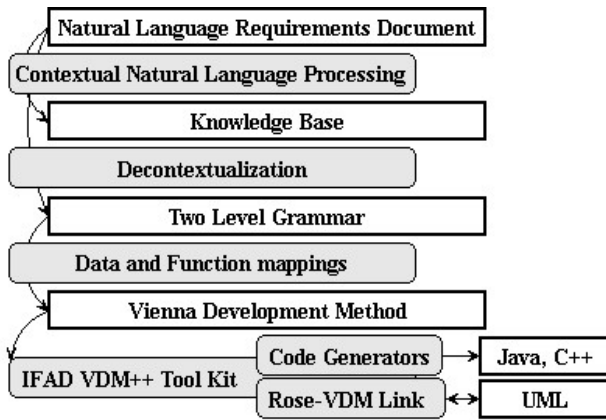
Figure 1: System Structure.

verted into TLG by removing the contextual dependency in the knowledge base. Finally the TLG code is translated into VDM by data and function mappings. This research is based upon the theories of NL processing, requirements analysis, formal specification, and programming languages. The entire system structure is shown in Figure 1.

2. EXAMPLE

In the sections which follow, we will present a simple example to illustrate our approach and describe the various system components. The following specification of an Automatic Teller Machine (ATM) will be used as the running example throughout the paper to illustrate the system. The details of how the bank verifies ID and PIN and how it updates the balance are omitted for the sake of simplicity.

```
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.
```

3. CONSTRUCTION OF KNOWLEDGE BASE FROM REQUIREMENTS

The knowledge base is built from a requirements document using syntactic, semantic and contextual (discourse) information. The knowledge base has to be constructed in a way to store the information without any data redundancy while facilitating ease in manipulating the data [9]. Also the knowledge representation has to capture the corresponding structure of TLG for the later translation.

First the requirements document is converted into Extensible Markup Language (XML) by dividing the document into sections, each section into subsections, each subsection

into paragraphs, and finally each paragraph into sentences. The formats of requirements documents can vary in their formats and styles. Therefore by converting them into XML the documents can have one standard format. Also by using XML, we can use the tables and diagrams information in the document, which are usually ignored for the systems that read out only the text from the document. The discourse level information such as sections, subsections, and paragraphs are used later when the contextual information is to be stored in the knowledge base. A part of ATM requirements document in XML is shown as follows.

```
<p>
  <s>ATM has 3 service types; withdraw, deposit,
     and balance check.</s>
  <s>For each service first it verifies ID and PIN
     from the bank giving the balance.</s>
</p>
<p>
  <s>ATM withdraws an amount with ID and PIN giving
     the balance in the following sequence.</s>
  <s>If the amount is less than or equal to the
     balance then it decreases the balance by the
     amount.</s>
  <s>And then it updates the balance in the bank
     with ID.</s>
</p>
  .....
```

'p' and 's' stand for paragraph and sentence respectively.

Once the document is formatted in XML, each sentence is read by the system and each sentence is parsed 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, object, complement) of each word is determined by top-down parsing [2]. Separating the parsing process into these two different sub-processes, a unique approach compared with most other parsing techniques, 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 [16]. 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 part of speeches (frequently used ones being listed first) of about 85000 words from [11] are used to resolve the syntactic ambiguity in this phase. Also 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)
```
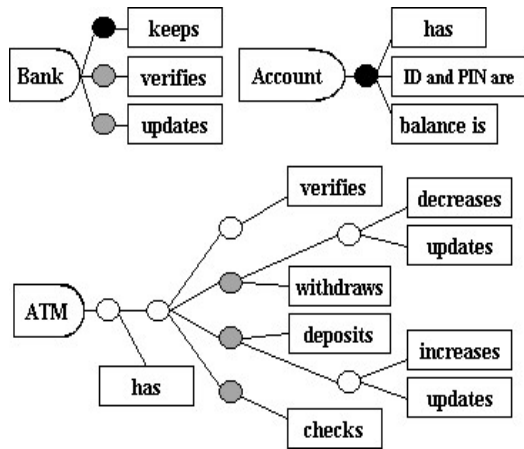
Figure 2: Knowledge base for ATM.

```
Part of sentence : ( subject verb adjective object
 adverb preposition_object )
```

Using semantic information and syntactical information gathered from the previous phase, anaphoric references (pronouns) are identified. 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) [5, 12].

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 the contextual 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 from what is originally meant. 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) in XML and semantics information in sentences. For instance, in the ATM example the phrase "in the following sequence" indicates that the following sentences are likely to stay in the same context as the current context. Alternatively a sub-context to hold the following sentences has to be created under the current context. The contextual structure of the knowledge base is shown in the Figure 2. Only the main verb for each sentence is shown in a rectangle due to the space limitation. 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 the location of the sentence "For each service first it verifies ID and PIN from the bank giving the balance". It is positioned under the same context with the withdraw, deposit, and balance check service operations, to be used for each service later. If it were located in any other context, the sentence couldn't operate as originally intended.

The knowledge base is constructed in XML. This enables the user to see the structured context information of the system. Because both the context in the knowledge base and XML data have the tree-like structure, the knowledge base in XML visualizes the same structure of the knowledge base shown in Figure 2. A part of the knowledge base in XML is shown below.

```
<c name = "ATM">
 <c>
  <s>ATM has 3 service types ; withdraw, deposit,
   balance check</s>
  <c>
   <s>first ATM verifies ID and PIN from bank for
   each service giving balance</s>
  </c>
  <c>
   <s>ATM withdraws amount with ID and PIN giving
   balance in following sequence</s>
   <c>
    <s>if amount is less than or equal to balance
    then ATM decreases balance by amount</s>
    <s>then ATM updates balance in bank with ID</s>
   </c>
   ......
  </c>
 </c>
</c>
```

where 'c' and 's' stand for context and sentence respectively.

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, the syntactic, semantic, and contextual information is used to build up the contextual knowledge base from the requirements documentation.

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. 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 [6]. 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 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 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.
```

When the system proceeds with the ATM knowledge base, it detects the fact that the data type of the `Amount` hasn't been explicitly specified, although it is likely to be `Float` since it is used in arithmetic with `Balance`. So it asks for verification 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 (:=). Therefore the systematic structure of the knowledge base and the formal yet flexible syntax of TLG make very smooth translation from the former to the latter.

5. TRANSLATION FROM TLG TO VDM

The object-oriented extension of the Vienna Development Method meta-language, VDM++ [8], has been selected 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 [13]. A detailed description of the translation from TLG to VDM++ is given in [7]. The TLG specification of the ATM example would be translated into the following VDM++ specfication.

```
class Bank

instance variables
  private Accounts_List : seq of Account := []

operations
  public verify : int * int ==> real
  verify (ID, PIN) ==
    (dcl Balance : real := 0;
    return Balance);

  public update : real * int ==> ()
  update (Balance, ID) ==
    return

end Bank

class Account

instance variables
  private ID : int ;
  private PIN : int ;
  private Balance : real

end Account

class ATM

instance variables
  private CBank : Bank := new Bank()

operations
  public withdraw : real * int * int ==> real
  withdraw (Amount, ID, PIN) ==
    (dcl Balance : real;
    Balance := CBank.verify(ID, PIN);
    if Amount <= Balance then
      (Balance := ( Balance - Amount ) ;
      CBank.update(Balance, ID));
    return Balance);

  public deposit : real * int * int ==> real
  deposit (Amount, ID, PIN) ==
    (dcl Balance : real;
    Balance := CBank.verify(ID, PIN);
    Balance := ( Balance + Amount );
    CBank.update(Balance, ID);
    return Balance);

  public checkBalance : int * int ==> real
  checkBalance (ID, PIN) ==
    (dcl Balance : real;
    Balance := CBank.verify(ID, PIN);
    return Balance)

 end ATM
```

Once we have translated the TLG specification into a VDM++ specification we can convert this into a high level language such as Java^{TM} or C++, using the code generator that the VDM Toolkit^{TM} provides [14]. 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) [18] using a link with Rational Rose^{TM} [15].

6. 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. 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.

Using the formalized context in NL processing to handle the ambiguity in NL and TLG with its NL-like syntax and flexibility without losing its formalism to build a bridge between two different formalisms of NL and a formal specification language, we can achieve fully automated conversion from an NL requirements document to a formal specification language for the prototyping and implementation of complex systems. Also since we use the requirements document itself as the input for the system generation, the dependency of on-line comments or extra documents to describe the system becomes small.

By using XML to format the requirements document, the input document becomes standardized possibly making uses of tables and diagrams in the document as well. Also the knowledge base can be queried using natural language and displayed in XML format with the contextual information.

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 describing 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 natural language processing 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.

7. REFERENCES

[1] V. S. Alagar and K. Periyasamy. *Specification of Software Systems*. Springer-Verlag, 1998.

[2] J. Allen. *Natural Language Understanding*. Benjamin/Cummings, 2nd edition, 1995.

[3] D. Bjørner and C. B. Jones. *The Vienna Development Method: The Meta-Language*. Springer-Verlag, 1985.

[4] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. Technical report, W3C Recommendation REC-xml-19980210, 1998.

[5] S. Brennan, L. Friedman, and C. Pollard. A Centering Approach to Pronouns. *Proc. 25th ACL Annual Meeting*, pages 155–162, 1987.

[6] B. R. Bryant. Object-Oriented Natural Language Requirements Specification. *Proc. ACSC 2000, 23rd Australasian Comp. Sci. Conf.*, pages 24–30, 2000.

[7] B. R. Bryant and B.-S. Lee. Two-Level Grammar as an Object-Oriented Requirements Specification Language. *Proc. 35th Hawaii Int. Conf. System Sciences (to appear)*, Jan. 2002.

[8] E. H. Dürr and J. van Katwijk. VDM++ - A Formal Specification Language for Object-Oriented Designs. *Proc. TOOLS USA '92, 1992 Technology of Object-Oriented Languages and Systems USA Conf.*, pages 63–278, 1992.

[9] D. Fensel, R. Groenboom, and G. R. Renardel. Modal Change Logic (MCL): Specifying the Reasoning of Knowledge-based Systems. *Data and Knowledge Engineering*, 26:243–269, 1998.

[10] N. E. Fuchs and R. Schwitter. Attempto Controlled English (ACE). *Proc. CLAW 96, 1st Int. Workshop Controlled Language Applications*, 1996.

[11] W. Grady. Moby Part-of-Speech II (data file), 1994.

[12] B. J. Grosz, A. K. Joshi, and S. Weinstein. Providing a Unified Account of Definite Noun Phrases in Discourse. *Proc. 25th ACL Annual Meeting*, 155-162:44–50, 1983.

[13] IFAD. The VDM++ Toolbox User Manual. Technical report, IFAD (http://www.ifad.dk), 2000.

[14] IFAD. VDMTools - Java/C++ Code Generator. Technical report, IFAD, 2000.

[15] IFAD. VDMTools - The Rose-VDM++ Link. Technical report, IFAD, 2000.

[16] D. Jurafsky and J. Martin. *Speech and Language Processing*. Prentice Hall, 2000.

[17] J. McCarthy. Notes On Formalizing Context. Technical report, Computer Science Department. Stanford University. Stanford, CA, 1993.

[18] T. Quatrani. *Visual Modeling with Rational Rose 2000 and UML*. Addison-Wesley, 2000.

[19] A. van Wijngaarden. *Orthogonal Design and Description of a Formal Language*. Mathematisch Centrum, Amsterdam, 1965.

[20] W. M. Wilson. Writing Effective Natural Language Requirements Specifications. Technical report, Naval Research Laboratory, 1999.