Semantic Analysis of Software Constraints

Imran Sarwar Bajwa, Mark Lee, Behzad Bordbar
School of Computer Science, University of Birmingham, B15 2TT, Birmingham, UK
{i.s.bajwa, m.g.lee, b.bordbar}@cs.bham.ac.uk

Abstract

In this paper, we present a novel approach NL2OCL to translate English specification of constraints to formal constraints such as OCL (Object Constraint language). In the used approach, input English constraints are syntactically and semantically analyzed to generate a SBVR (Semantics of Business Vocabulary and Rules) based logical representation that is finally mapped to OCL. During the syntactic and semantic analysis we have also addressed various syntactic and semantic ambiguities that make the presented approach robust. The presented approach is implemented in Java as a proof of concept. A case study has also been solved by using our tool to evaluate the accuracy of the presented approach. The results of evaluation are also compared to the pattern based approach to highlight the significance of the used approach.

Introduction

In the recent years, a few research contributions have been presented in the area of automatic translation of natural language (NL) specifications to formal specifications such as UML (OMG, 2007) (Unified Modeling Language) class models (Harmain, 2003), Java (Price and Riloff, 2000), and SQL (Structured Query Language) queries (Giordani, 2008). However, the available tools are limited to 65%-70% levels of accuracy in real time software development. Researchers have shown that the key reason of less accuracy is the various types of syntactic and semantic ambiguities (Kiyavitskaya, 2008) of the natural languages. For example, Mich (2004) showed that 71.8% of a sample of NL software specification is ambiguous. Hence, the ambiguous and incomplete specifications lead to inconsistent and absurd formal specifications such the software models or the software constraints.

In this paper, we present a novel approach to increase OCL (OMG, 2010) acceptance in software designers and developers community by simplifying the generation of OCL from English specification. However, the key challenge in generation of OCL from English specification was to overcome the inherent syntactic and semantic ambiguities (Uejima, 2003) in English. The NL2OCL approach works as the input English specification of constraints is first syntactically and semantically analyzed and then a SBVR (OMG, 2008) based logical representation is generated. A SBVR based representation is easy to machine process and easy to translate to other formal languages such as OCL due to its foundation on higher order logic (formal logic).

The remaining paper is structured into the following sections: Section 2 illustrates the architecture of NL2OCL approach. Section 3 presents a case study used to evaluate the presented approach. Finally, the paper is concluded to discuss the future work.

Semantic Analysis of English Constraints

For translating English specification of constraints to OCL constraints, the NL2OCL approach was used. In NL2OCL approach, two inputs are given: a txt file containing English specification of a constraint, and a UML class model in EMF (Eclipse Modeling Framework) ECORE format. First English specification is syntactically and semantically analyzed to extract OCL elements and then finally an OCL expression is generated (see figure 1):

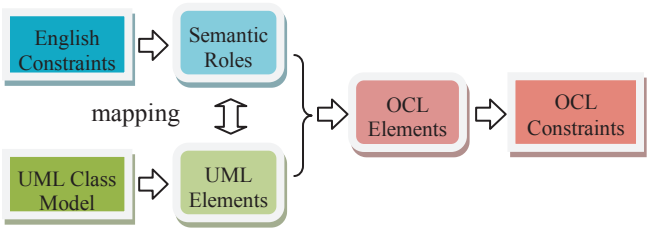

Figure 1. English to OCL Translation Approach

Processing English Specification of Constraints

The processing of input English (constraints) text is started with the preprocessing. Afterwards, the pre-processed English text is syntactically analyzed to identify the syntactic structures and dependencies among them. Output of the syntactic analysis is further semantically analyzed to generate a logical representation.

Preprocessing English Text

In the preprocessing phase, the input English text is preprocessed before deep processing. Following steps are involved in preprocessing:

Part-of-Speech (POS) Tagging: In this step, the input English text is tokenized and part-of-speech is identified for each token. The Stanford POS tagger (Toutanova, 2003) version 3.0.3 has been used to identify 44 various tags. The Stanford POS tagger is 97% (Manning, 2011) accurate. However, in a few cases, the Stanford POS tagger wrongly tags English words. For example in Figure 3, token 'books' is identified as a noun but the token 'books' is verb and should be tagged as 'VBZ' instead of 'NNS'.

English: A customer books two items.

Tagging: [A/DT] [customer /NN] [books/NNS] [two/CD] [items/NNS] [./.]

Figure 2. Part-of-Speech tagged text

This problem becomes more serious as we are using the Stanford Parser for syntactic analysis, and the POS tagging goes wrong rest of the parsing (such as parse tree and typed dependencies) by the Stanford parser goes wrong.

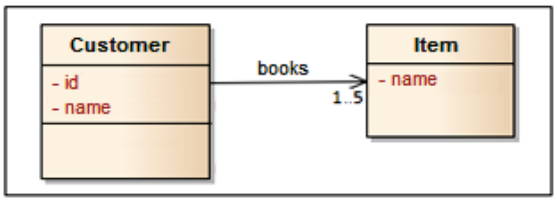

Figure 3. A UML Class model

We have addressed such cases by mapping all the tokens to the target UML model to confirm the tags. Example discussed in Figure 2 can be easily solved by mapping all the tags to the available information in UML class model of Figure 3. It is given in Figure 3 that, 'books' is a relationship and it should be tagged verb or 'VBZ' by using the mappings given in Table 1.

Class model elements	English language elements
Class names	Common Nouns
Object names	Proper Nouns
Attribute names	Generative Nouns, Adjectives
Method names	Action Verbs
Associations	Action Verbs

Table 1: Mapping UML class model to English

Lemmatization: In lemmatization phase, the inflectional endings are removed and the base or dictionary form of a word is extracted, which is known as the lemma. We identify lemma (base form) of the tokens (all nouns and verbs) by removing various suffixes attached to the nouns and verbs e.g. in Figure 3, a verb "awarded" is analyzed as "award+ed". Similarly, the noun "workers" is analyzed as "worker+ s". Here, 's' is helpful in identifying that person is plural.

Syntactic Analysis

We have used the Stanford parser to parse the pre-processed English text. The Stanford parser is 84.1% accurate (Cer, 2010). However, the Stanford parser is not capable of voice-classification. Hence, we have developed a small rule-based module classifies the voice in English sentences. In syntax analysis phase, three steps are performed as below:

Generating Syntax Tree: The Stanford parser is used to generate parse tree and typed dependencies (Marneffe, 2006) from POS tagged text. However, there are some cases where the Stanford parser cannot identify correct dependencies. For example, in Figure 4, the Stanford Parser wrongly associates 'employees' with 'bonus'. Whereas, that correct dependency should be `prep_with(card-3, credit-11)` to represent the actual meanings of the example i.e. the credit cards with free credit are given to the customers.

English: The pay is given to all employees with bonus.

Typed Dependency (Collapsed :

```
det(pay-2, The-1)
nsubjpass(given-4, pay-2)
auxpass(given-4, is-3)
det(employees-7, all-6)
prep_to(given-4, employees-7)
prep_with(employees-7, bonus-9)
```

Figure 4. Typed Dependencies by the Stanford parser

To handle such inaccurate dependencies, we need the context of this statement such as a UML class model is the context for a constraint. Hence, we can use the UML class model shown in figure 5 to correct dependencies. The relationships in UML class model such as the associations (directed and un-directed) can be used to deal with syntactic ambiguities such as attachment ambiguity (Kiyavitskaya, 2008). For example in Figure 6, it is shown that 'Bonus' is associated to 'Pay' and there is no association in 'Employee' and 'Bonus' classes. By using

this class association, we can correct the dependency as `prep with(card-3, credit-11)` instead of `prep_with(customer-8, credit-11)`.

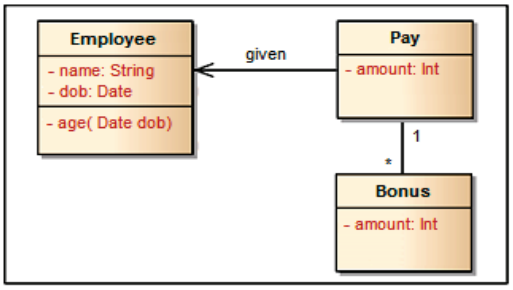

Figure 5. A UML class model

Voice Classification: In voice classification phase, the sentence is classified into active or passive voice category. We have used our rule-based module to identify the voice of a sentence as the Stanford parser does not provide this facility. The used rules for voice classification are based on grammatical structure of the English sentence. Various grammatical features manifest passive-voice representation such as the use of past participle tense with main verbs can be used for the identification of a passive-voice sentence. Similarly, the use of 'by' preposition in the object part is also another sign of a passive-voice sentence. However, the use of by is optional in passive-voice sentences.

Intermediate Representation: Outcome of the syntactic analysis phase is an intermediate representation. A tabular representation is generated containing the syntactic chunks and their associated representation such as syntax type (such as a subject, a verb or an object), quantification, dependency, and associated preposition.

#	Chunk	Syntax	Quat.	Depen.	Prep.	EOS
1	Pay	Subject	the	given		
2	is given	Verb		is		
3	employees	Object	all	given	to	
4	Bonus	Adverb		pay	with	True

Table 2. An intermediary representation

A major feature of this intermediary representation is that the active-voice and passive-voice are mapped to same representation such as subject of a passive-voice sentence is represented as object and object of a passive-voice sentence is represented as subject.

Semantic Analysis

In semantic analysis phase, we aim to understand the exact meanings of the input English text; to identify the relationships in various chunks and generate a logical representation. For semantic analysis English constraints, we have to analyze the text in respect of particular context such as UML class model. Our semantic analyzer performs following three steps to identify relations in various syntactic structures:

Shallow Semantic Parsing: In shallow semantic parsing, the semantic or thematic roles are typically assigned to each syntactic structure in a English sentence. Semantic labeling on a substring (semantic predicate or a semantic argument) in a constraint (English sentence) 'S' can be applied. Every substring 's' can be represented by a set of words indices:

$$S \subseteq \{1, 2, 3, \ldots, n\}$$

Formally, the process of semantic role labeling is mapping from a set of substrings from c to the label set 'L'. Where L is a set of all argument semantic labels:

$$L = \{a_1, a_2, a_3, \ldots, m\}$$

We use SBVR vocabulary as the target semantic roles due to the fact that the mapping of SBVR vocabulary to OCL is easy and straightforward. We have identified mappings of English text elements to SBVR vocabulary (see Table 3).

English Text elements	SBVR Vocabulary
Common Nouns	Object Type
Proper Nouns	Individual Concept
Generative Noun, Adjective	Characteristic
Action Verbs	Verb Concepts
Subject + verb + Object	Fact Type

Table 3: Mapping class model to English

Following are the three main steps involved in the phase of semantic role labelling of English constraints:

a. Identifying the Predicates: In first step, system identifies the words in the sentence that can be semantic predicates or semantic arguments. We have identified predicates in following two phases:

Step I-. In English, predicates can be in the form of a simple verb, a phrasal verb or a verbal collocation.

Step II- Predicate arguments are typically nouns in subject and object part of a sentence. In English, nouns can have pre-modifiers such as articles (determiners) and can also have post-modifiers such as prepositional phrases, relative (finite and non-finite) clauses, and adjective phrases.

b. Sense Recognition: After a predicate is identified, we need to recognize the exact sense of the predicates so that accurate semantic roles may be assigned to the predicate. Sense recognition at this phase is important as some NL elements can be ambiguous e.g. a verb can be assigned the semantic role of 'Verb Concept' or a 'Fact Type'. Such information is mapped to navigation expressions in OCL. We can identify correct semantic role be mapping information to the UML class model by checking that verb

is an operation or an association. If a verb is operation it is mapped to 'Verb Concept' else it is mapped to a 'Fact Type'.

c. Assigning the Thematic Roles: Once the predicates are identified, semantic roles are assigned by using the mappings given in Table 2. Role classification is performed as the syntactic information (part of speech and syntactic dependencies). The output of this phase is semantic roles assigned to the predicates and the predicate arguments (see Figure 6).

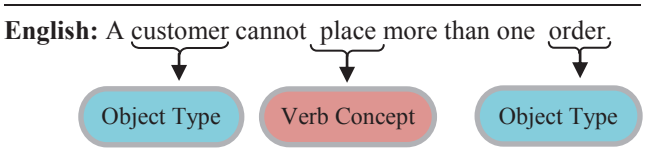

Figure 6. Semantic roles assigned to input English sentence

Deep Semantic Analysis: The computational semantics aim at grasping the entire meanings of a natural language sentence, rather than focusing on text portions only. For computational semantics, we need to analyze the deep semantics of the input English text. The deep semantic analysis involves generation of a fine-grained semantic representation from the input text. Various aspects are involved in deep semantics analysis. However, we are interested in quantification resolution (see Figure 7) and quantifier scope resolution:

a. Resolving Quantifications: In English constraints, the quantifiers are most commonly used. We not only cover all two traditional types (Universal and Existential) of quantifications in FOL but also we have used two other types: Uniqueness and Solution quantification. Following are the details of identifying various quantifications in English constraints.

i. Universal Quantification (∀X): In English, the quantification structures such as 'each', 'all', and 'every' are mapped to universal quantification. Similarly, the determiners 'a' and 'an' used with the subject part of the sentence are treated as universal quantification due to the fact that we are processing constraints and generally constraints are mentioned for all the possible X in a universe.

ii. Existential quantification (∃X): The keywords like many, little, bit, a bit, few, a few, several, lot, many, much, more, some, etc are mapped to existential quantification.

iii. Uniqueness Quantification (∃=1X): The determiners 'a' and 'an' used with object part of the sentence are treated as uniqueness quantification.

iv. Solution Quantification (§X): If the keywords like more than or greater than are used with n then solution quantifier is mapped to At-most Quantification. Similarly, if the keywords like less than or smaller than are used with *n* then solution quantifier is mapped to At-least *n* Quantification.

b. Quantifier Scope Resolution: After identifying the quantifications, we also need to resolve the scope of quantifiers in input English text. For quantification variable scoping, we have treated syntactic structures as logical entities. Moreover, the multiplicity given in the target UML class model also helps in identifying a particular type of quantification. For example, in figure 5, the multiplicity '1' specifies that customer can get at most one credit card. This will be equal to At-most n quantification in SBVR.

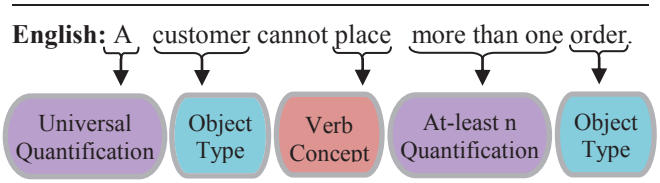

Figure 7. Semantic roles assigned to input English sentence

Semantic Interpretation: After shallow and deep semantic parsing, a final semantic interpretation is generated that is mapped to SBVR and OCL in later stages. A simple interpreter was written that uses the extracted semantic information and assigns an interpretation to a piece of text by placing its contents in a pattern known independently of the text. Figure 8 shows an example of the semantic interpretation we have used in the NL2OCL approach:

English: A customer can place one order.

Semantic Interpretation:

```
( place
   (object_type = (∀X ~ (customer ? X)))
      (object_type = (∃ =1Y ~ (order ? Y))))
```

Figure 8. Semantic roles assigned to English sentence.

Mapping Logical Form to OCL

Once we get the logical representation of English constraint, it is mapped to the OCL by using model transformation technology. For model transformation of NL to OCL, we need following two things to generate OCL constraints:

i. Select the appropriate OCL template (such as invariant, pre/post conditions, collections, etc)
ii. Use set of mappings that can map source elements of logical form to the equivalent elements in used OCL templates.

We have designed generic templates for common OCL expressions such as OCL invariant, OCL pre-condition, and OCL post-condition. User has to select one of these three templates manually. Once the user selects one of the constraints, the missed elements in the template are

extracted from the logical representation of English constraint. Following is the template for invariant:

```
package [UML-Package]
context [Object-Type]
inv: [Body]
```

Following is the template we used for OCL pre-condition:

```
package [UML-Package]
context [Object-Type : :Verb_Concept(Characteristic): Return_Type]
pre: [Body]
```

Following is the template we used for OCL post-condition:

```
package [UML-Package]
context [Object-Type : :Verb_Concept(Characteristic): Return_Type]
post: [Body]
Result: [Body]          -- optional
```

In the all above shown templates, elements written in brackets '[ ]' are required. We get these elements from the logical representation of English sentence. Following mappings are used to extract these elements:

i. *UML-Package* is package name of the target UML class model.

ii. *Object-Type* is name of the class in the target UML Class model and the Object Type should also be in the subject part of the English Constraint.

iii. *Verb_Concept* is one of the operations of the target class (such as context) in the UML Class model.

iv. *Characteristic* is the list of input parameters of a *Class* and we get them from the UML class model.

v. *Return-Type* is the return data type of the *Object-Type* and we get them from the UML class model. The return type is the data-type of the used Characteristic in English constraint and this data type is extracted from the UML class model.

vi. *Body* can be a single expression or combination of more than one expression. *Body* is generated using mappings similar to given in Table 4, 5, 6, and 7. For complete mappings, reader should consult (Bajwa, 2011).

Case Study

In this section, we present a case study on the "Royal & Loyal" model. The Royal & Loyal model was originally presented for introducing *OCL By Example* in (Warmer and Kleppe, 2003). Afterwards, the Royal & Loyal model is used in various publications, e.g., (Tedjasukmana, 2006, Dzidek et al., 2005, Wahler, 2008). The same model is also shipped with several tools as an example model, e.g., (Dresden Technical University, 2007).

The Royal & Loyal Model Constraints

The Royal & Loyal case study has also been solved by Wahler (2008) in his PhD thesis. We aim to compare the results of our approach to Pattern based approach as Wahler's approach is the only work that can generate OCL constraints from a natural language. There are 26 English constraints in the Royal & Loyal case study. Wahler solved 18 English constraints into OCL out of 26 using his (pattern-based) approach. In comparison to Wahler's pattern based approach, our NL-based approach has successfully translated 25 constraints to OCL. In the following section, we present three examples of constrains due to shortage of space.

Example I

English: The owner of a customer card must participate in at least one loyalty program.

SBVR: It is necessary that the *owner* of a **customercard** *must participate* in at least one **loyaltyprogram**.

OCL:
```
package: royal_and_loyal
context CustomerCard
inv self.owner.programs ->
Size()>= 1
```

Other OCL:
```
context CustomerCard
    inv programParticipation: self
.owner.programs ->size() > 0
```

Example II

English: There must be at least one transaction for a customer card with at least 100 points.

SBVR: It is necessary that there *must be* at least one **transaction** for a **customercard** with at least 100 *points*.

OCL:
```
package: royal_and_loyal
context CustomerCard
inv self.transaction-
>select(point >= 100)->Size()>=1
```

Other OCL:
```
context CustomerCard
    inv transactionPoints : self
 .transactions->
select(points>100) -> notEmpty()
```

Example III

English: The service level of each membership must be a service level known to the loyalty program.

SBVR: It is necessary that **servicelevel** of each **membership** *must be* a **servicelevel** *known* to **loyaltyprogram**.

OCL:
```
package: royal_and_loyal
context Membership
inv self.currentLevel.levels ->
includes(programs)
```

Other OCL:
```
context Membership
inv knownServiceLevel:
programs.levels ->
includes(currentLevel)
```

There was only one constraint that was not fully translated by our NL base approach due to the limitation that the vocabulary used in English constraint should also be part of the input UML class model. In the following English constraint, concepts 'credits' and 'debits' are not part of the Royal & Loyal model (Warmer and Kleppe, 2003: pp.22).

*If none of the services offered in a loyalty program **credits** or **debits** the loyalty accounts, then these instances are useless and should not be present.*

In comparison of both approaches (see Table 8), NL-based approach produced for better results than the pattern based appoach:

Approach Type	Total Constraints	Solved Constraints	Percentage
Pattern based Approach	26	18	69.23%
NL Based Approach	26	25	96.13%

Table 8. Pattern based Approach vs NL Based Approach

Another advantage over Wahler's approach is that our NL-based approach is fully automatic, while in Wahler's pattern based approach, user has to do detailed manual analysis of the English constraints to choose the right pattern and then Wahler's tool Copacabana (Wahler, 2008) translates the pattern instances to OCL code.

Conclusion and Future Work

The current presented work focuses on automated (object oriented) analysis of NL specification and generation of OCL constraints for UML models. The presented work not only complements the current research work in the field of automated software modeling but also simplifies the process of writing OCL constraints. The initial performance evaluation of our approach is very encouraging and symbols the efficacy. The Software modelers can get benefit of our tool as the NL2OCL can generate accurate OCL constraints with less effort.

However, our tool is limited to process one English constraint (sentence) at a moment. In future, we aim to enhance our tool to process multiple constraints.

References

Bajwa, I.S., Lee, M.G. 2011. Transformation Rules for Translating Business Rules to OCL Constraints. in 7th European Conference on Modelling Foundations and Applications(ECMFA 2011). Birmingham, UK. Jun 2011. pp:132-143

Chen, B., Su J., and Tan, C.L. 2010. Resolving Event Noun Phrases to Their Verbal Mentions, in Empirical Methods in Natural Language Processing, Pages 872-881, Cambridge, MA, October, 2010

Cer, D., Marneffe, M.C., Jurafsky, D. and Manning, C.D. (2010). Parsing to Stanford Dependencies: Trade-offs between speed and accuracy." InProceedings of LREC-10.

Giordani A. 2008. Mapping Natural Language into SQL in a NLIDB, Natural Language and Information Systems, 2008, Volume 5039/2008, 367-371

Harmain, H. M., Gaizauskas R. 2003. CM-Builder: A Natural Language-Based CASE Tool for Object- Oriented Analysis. Automated Software Engineering. 10(2):157-181.

Warmer, J.B. and Kleppe, A.G. 2003.The object constraint language: getting your models ready for MDA. Second Editotion, Addison Wesley

Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D. (2008). Requirements for tools for ambiguity identification and measurement in natural language requirements specifications, Requirements Engineering, Vol. 13, No. 3. (2008), pp. 207-239.

Manning, C.D. (2011). Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? In proceedings of CICLing (1) 2011. pp.171~189

Marneffe, M.C., MacCartney Bill and Manning, C.D. (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In LREC 2006.

Mich, L., Franch, M., Inverardi, P.N.: Market research for requirements analysis using linguistic tools. Requir. Eng.(2004) pp.40-56

OMG. 2007. Unified Modeling Language (UML), OMG Standard, v. 2.3.

OMG. 2008. Semantics of Business Vocabulary and Rules (SBVR), OMG Standard, v. 1.0.

OMG. 2010. Object Constraint Langauge (OCL), OMG Standard, v. 2.2.

Price, D., Riloff, E., Zachary, J., and Harvey, B. (2000) "NaturalJava: A Natural Language Interface for Programming in Java", In Proceedings International Conference on Intelligent User Interfaces (IUI) 2000.

Toutanova K., Klein D., Manning C., and Singer Y. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003, pp. 252-259.

Uejima, H. , Miura, T., Shioya, I. (2003). Improving text categorization by resolving semantic ambiguity Communications, Computers and signal Processing, 2003 pp. 796-799

Wahler M. 2008. Using Patterns to Develop Consistent Design Constraints. PhD Thesis, ETH Zurich, Switzerland, (2008)