Automated Generation of OCL Constraints: NL based Approach vs Pattern Based Approach

IMRAN SARWAR BAJWA*, AND MUHAMMAD ANWAR SHAHZADA**

RECEIVED ON 02.09.2015 ACCEPTED ON 11.05.2016

ABSTRACT

This paper presents an approach used for automated generations of software constraints. In this model, the SBVR (Semantics of Business Vocabulary and Rules) based semi-formal representation is obtained from the syntactic and semantic analysis of a NL (Natural Language) (such as English) sentence. A SBVR representation is easy to translate to other formal languages as SBVR is based on higher-order logic like other formal languages such as OCL (Object Constraint Language). The proposed model endows with a systematic and powerful system of incorporating NL knowledge on the formal languages. A prototype is constructed in Java (an Eclipse plug-in) as a proof of the concept. The performance was tested for a few sample texts taken from existing research thesis reports and books.

Key Words: Object Constraint Language, Constraints, Natural Language Processing.

1. INTRODUCTION

The UML (Unified Modeling Language) [1] is now widely considered as a de facto standard family of languages for specifying, modelling, constructing and documenting object-oriented software and systems. Popularity of UML is often attributed to its semi-formal nature. It is argued that UML is not formal enough to demand deep knowledge of formal methods that inhibits practically minded software engineers from using it. As a result, usability is seen as a major feature of the UML. However, OCL [2], which is one of the languages in UML, is a clear exception to this argument. OCL plays a key role in UML modelling for expressing essential constraints to make UML models well-defined. But it is also a common knowledge that OCL is the least adopted amongst all languages in UML [3].

We have identified three major factors contributing to usability problems in OCL. The primary factor is the hard syntax of OCL [4]. Wahler [3] addressed this problem by introducing a template based language. His approach, which is implemented to be used with IBM Rational, allows the user to pick a template, from a wide range of OCL template, assign the parameters and use them. This would greatly help the user; however, the key challenge is to learn which template to pick. Second aspect of OCL's usability problem is the ambiguous nature of OCL constraint as several equivalent implementations for a constraint are possible in OCL [5-6]. Cabot proposed an approach for automatic disambiguation of the constraints by means of providing a default interpretation for each kind of ambiguous expression. But a designer has to be

* Department of Computer Science, The Islamia University of Bahawalpur, Bahawalpur.
** Department of Computer Science, National College of Business Administration & Economics, Lahore.

aware of all the possible states while writing an OCL constraint to avoid the identified ambiguities. Third aspect of OCL's usability problem is understandability of overly complex OCL expressions commonly used in large software models [7]. The refactoring techniques are used to improve the understandability of OCL specifications but the employment of refactoring technique can be an overhead in the process of software modelling.

To contribute to OCL's usability a tool has to be able to deal with English. In practice, an English expression of a constraint is manually mapped to an OCL constraint on a given UML model. We have identified a set of tasks that are involved in typical English to OCL mapping. Firstly, since OCL is side-effect free [2], the English statement must be about the system, i.e. the terms and vocabulary used must be already existence in the model. Secondly, English language is inherently ambiguous. It is important to start from a correct understanding of the meaning of the expression. Thirdly, we believe that if the English sentence is clear and well-understood, the creation of OCL can be automated. On the basis of the above three points, this paper presents a framework for automated creation of OCL statements from the English language expressions.

Our approach allows the user to write various constraints and pre/post conditions on a UML model in English. First input the English is mapped with input UML model. Then the English constraints are automatically transformed to the equivalent OCL expressions via SBVR [8]. SBVR is an OMG's recent standard that and we have used SBVR to overcome the inherent ambiguity of English language. SBVR not only provides English a semantically formal representation but also closed to OCL syntax as both languages are based on formal logic. To create an OCL expression, the SBVR rules are transformed to OCL using MDA model transformations. As a proof of concept, the proposed approach is implemented as an Eclipse plugin called NL2OCLviaSBVR. The NL2OCLviaSBVR automatically transforms English to OCL via SBVR. The automated transformation not only hides the complexity involved in the manual production of OCL constraints from English language but also results in producing OCL constraints in a seamless and non-intrusive manner.

The rest of the paper is structured as follows: section 2 describes the NL-based software tool NL2OCL via SBVR; section 3 discuses a case study; section 4 presents evaluation followed by the related work section. The paper ends with a conclusion section.

2. RELATED WORK

In the last couple of decades, various research efforts are introduced in the domain of machine interpretation of English language description of software details to formal textual and graphical notations. A few examples of such works are given in Table 1.

TABLE 1. EXISTING WORK ON SOFTWARE REQUIREMENTS INTERPRETATION

No.	Work	Source
1.	Natural language to Unified Modeling Language 92]	[1,2,4]
2.	Natural language to Object Constraint Language [10]	[3]
3.	Natural language to Java code	[5,7]
4.	Natural language to Structured Query Language	[8,11]
5.	Natural language to ontologies	[12]
6.	Natural language to business process models	[13]
7.	Natural language to formal hardware verification properties	[14]

Additionally, different contributions are finished to automate various processes an phases of software modelling with the help of automated transformations. MDD (Model Driven Development) [8] is one of the recent developments in model transformation technology and MDD has permitted creation of a model from another in an automatic manner. Examples of such work are transformations example OCL/UML to Alloy [15], SBVR to OCL [16], SBVR to UML [17], UML/OCL to SBVR [18], OCL to B [19], SBVR to SQL [20], etc. Such automated transformations has made easy and simple to reuse the existing information.

3. THE NL TO OCL TOOL

The NL2OCLviaSBVR is a modular NL-based software tool that generates OCL constraints with respect to a target UML model. It takes two inputs: a single English statement and a UML model. To process the input English text first it is linguistically analyzed. In linguistic analysis of the English text, the English text is POS (Parts-Of-Speech) tagged. Then a rule-based parser is used to further process the POS tagged information to extract basic SBVR elements e.g. noun concept, fact type, etc. Here, the SBVR vocabulary is mapped to a SBVR rule. Finally, to generate an OCL expression, the SBVR vocabulary is mapped to OCL syntax using the model transformation approach. The working of these steps has been stated in detail in the following section:

3.1 The Input Documents

NL2OCLviaSBVR takes two input documents: an English text document and a UML model document. The English text is taken as a plain text file containing only English constraint. Current version of the NL2OCLviaSBVR handles only one English constraint at a time. The given English text should be grammatically correct. UML model is taken as XMI 1.0 format. We used Enterprise Architect to create a UML model and export it in XMI 1.0 format.

3.2 NL to SBVR Transformation

The core of NL2OCLviaSBVR is a NLP module that consists of a number of processing units organized in a pipelined architecture. This NLP module is highly robust and is able to process complex English statements. The NLP system is used to lexically and syntactically process the English text and then perform semantic analysis to identify basic SBVR elements. The core system processes a text into three main processing stages.

3.2.1 Preprocessing

In the preprocessing phase, the input text containing the natural language specification of an OCL constraint for a UML class model is preprocessed for deep processing. Major steps involved in preprocessing phase are splitting the sentences, tokenization, and lemmatization. The preprocessing sub-phases are discussed below:

Sentence Splitting: In first step, the input English text is read and broken into sentences. During sentence splitting, the margins of a sentence are identified and each sentence is separately stored. Sentence splitting is performed using the Stanford parser.

Tokenization: After sentence splitting, each sentence is processed to identify tokens. Again Stanford parser is employed for efficient tokenization. An example is shown in Fig. 1.

English: A customer can place one order.
Tokens: [A] [customer] [can] [place] [one [order] [.]

FIG. 1. TOKENIZED TEXT USING STANFORD PARSER

Lemmatization: Here, the morphological analysis of words is performed to remove the inflectional endings and to return the base or dictionary form of a word, which is known as the lemma. We identify lemma (base form) in the POS tagged tokens by removing various suffixes attached to the nouns and verbs.

3.2.2 Syntactic Analysis

The output of a typical syntax analysis phase is a tree diagram or other textual representation. Our syntactic analyzer parses the preprocessed text by POS-tagging information and defining the syntactic units also called chunks. In syntax analysis phase, four steps are performed as following:

POS Tagging: In this step, parts of speech are identified for each token in the input text. In POS tagging, each token is classified to its respective parts-of-speech category by assigning a specific tag to teach token such as NN, VB, RB, MD, DT, etc. The Stanford POS tagger version 3.0.3 has been used to identify 44 various parts of speech. An example of a POS tagged sentence is shown in Fig. 2.

Generating Syntax Tree: We have used Stanford Parser to generate parse tree. The Stanford parser is a lexically driven probabilistic parser based on PCFG (Probabilistic Context-Free Grammars) (Fig. 3).

3.2.3 Semantic Analysis

A typical semantic analysis yields in a logical form of a sentence. Logical form is used to capture semantic meaning and depict this meaning independent of a particular context. The goal of semantic analysis is to understand the exact meanings of the input text and identify that relationship in various chunks.

Shallow Semantic Parsing: In shallow semantic parsing, the semantic or thematic roles are typically assigned to easy syntactic structure in a NL sentence. This process is also called SRL (Semantic Role Labeling). Semantic labeling on a substring (semantic predicate or a semantic argument) in a constraint (NL sentence) 'S' can be applied. Every substring 's' can be represented by a set of words indices as following:

S †" {1, 2, 3, …., 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,

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

The semantic roles can act as an intermediate representation in NL to SBVR translation. In the context of the targeted representation (SBVR rule representation), we have incorporated the following semantic roles. These semantic roles are typically used in semantic role labeling.

English: A customer can place one order.
Tokens: [A/DT] [customer/NN] [can/MD] [place/VB] [one/CD [order/NN] [./.]

FIG. 2. PARTS-OF-SPEECH TAGGED TEXT

English: A customer can place one order.	
Tokens: (ROOT (S (NP (DT A) (NN customer)) (VP (MD can) (VP (VB place) (NP (CD one) (NN order)))) (. .)))	Dependencies: det(customer-2, A-1) nsubj(place-4, customer-2) aux(place-4, can-3) root(ROOT-0, place-4) num(order-6, one-5) dobj(place-4, order-6)

FIG. 3. PARSE REPRESENTATION AND DEPENDENCIES

(a)	Object Type	→	Common nouns
(b)	Individual Concept	→	Proper nouns
(c)	Verb Concepts	→	Main Verb
(d)	Characteristics	→	Generative Phrases

A sequence of steps was performed for labeling semantic roles to respective semantic predicates. Following are the three main steps involved in the phase of semantic role labeling:

Extracting Semantic Predicates: In this phase, we extract the possible semantic predicates. This module relies mainly on the external resources, thus the elements in target UML Class models (class names, attributes, methods) are likely to be semantic predicates. The chunks not matching the elements of target UML Class model are not semantic predicates or semantic arguments. For extracting semantic predicates we check if the verb is a simple verb, a phrasal verb or a verbal collocation and locate the verb in (Fig. 4).

In English sentences, verb concepts are typically represented in combination of auxiliary verb and main verb (possibly following participle). However sometimes, there are only auxiliary verbs and no main verbs (Fig. 5).

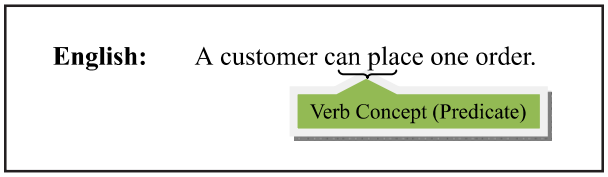


FIG. 4. IDENTIFYING VERB CONCEPTS (PREDICATE)

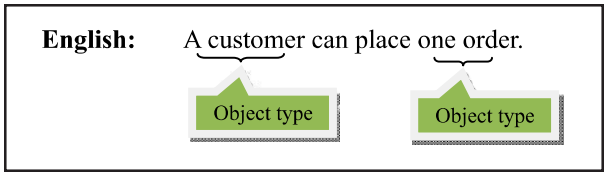


FIG. 5. IDENTIFYING SEMANTIC ARGUMENTS

Extracting Semantic Arguments: In English sentences, object type can be represented with pre-modifiers such as articles (determiners) and with post-modifiers: prepositional phrases, relative (finite and non-finite) clauses, and adjective phrases.

Semantic Interpretation: In lexical semantics, the frame is also considered a useful tool in text semantics and the semantics of grammar. The interpreter of a text invokes a frame when assigning an interpretation to a piece of text by placing its contents in a pattern known independently of the text. A text evokes a frame when a linguistic form or pattern is conventionally associated with that particular frame. Fig. 6 shows an example of the semantic interpretation we have used in the presented approach for NL to OCL transformation.

The output of the NLP module is an xml file that contains the parsed English text with all the extracted information.

Deep Semantic Parsing: In natural languages, quantifications are typically expressed with NPs (Noun Phrases). However, in FOL (First-Order Logic), the variables are quantified at the start of the logical expressions. Generally, the NL quantifiers are much more vague and varied. This vagueness makes translation of NL to FOL complex. However, we have set of heuristic rules to identify the quantifications:

(i) Universal Quantification (∀X): The universal quantification is mapped to Universal Quantification in SBVR. The NL quantification structures 'each', 'all', and

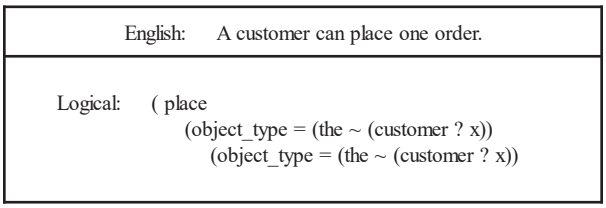


FIG. 6. SEMANTIC ROLES ASSIGNED TO INPUT ENGLISH SENTENCE.

'every' are mapped to universal quantificational structures. Similarly, the determiners 'a' and 'an' used with the subject part of the sentence are treated as universal quantification (Fig. 6).

(ii) Existential Quantification (∃X): The existential quantification is mapped to Existential Quantification in SBVR. 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 ($\exists_{=1}X$): The uniqueness quantification is mapped to Exactly-One Quantification in SBVR. The determiners 'a' and 'an' used with object part of the sentence are treated as uniqueness quantification.

(iv) Solution Quantification (§X): The solution quantification is mapped to Exactly-n Quantification in SBVR. If the keywords like more than or greater than are used with n then solution quantifier is mapped to At-most Quantification. Here, if the terms "less than" or "smaller than" are used with n then solution quantifier is mapped to At-least Quantification.

The SBVR produces a SBVR rule in the form of text string that is further formatted using the SBVR notation i.e. Structured English. The output SBVR module is saved and exported in two separate files: an xml file contains the SBVR vocabulary and respective details; a text file contains the formatted SBVR rule.

3.3 The SBVR to OCL Transformation

The OCL module maps a SBVR rule to an OCL expression by using model transformation that incorporates the mapping rules between SBVR and OCL metamodels. SBVR to OCL mapping rules typically define the conversion of element(s) of the SBVR metamodel to equivalent element(s) of the OCL metamodel. In OCL module, SiTra [21] library is used to implement model transformation. OCL module uses the output of the SBVR module i.e. the SBVR vocabulary to generate an OCL expression. A set of mapping rules were defined to map the SBVR vocabulary to different type of OCL expressions e.g. invariant, precondition and post condition. OCL queries have not been supported by the current version of the NL2OCLviaSBVR. A brief description of the mapping rules is provided in the following section:

3.3.1 OCL Package and Context

For any type of OCL expression, two elements are basic requirements: package and context. The UML package is mapped to the OCL package. While, the context of an OCL expression defines the scope of the given invariant or pre/post condition. To specify the context of an OCL invariant, the major actor in the SBVR rule is extracted to specify the context. To specify the context of an OCL pre/post condition, the action performed by the actor in a SBVR rule is considered as the context.

3.3.2 Mapping OCL Constraints

Transformation rules for mapping of UML-SBVR specification to OCL constraints are defined in this section. There are two basic types of an OCL constraints; invariant of a class, and pre/post condition of an operation. Constraint on a class is a restriction or limitation on a particular attribute, operation or association of that class with any other class in a model [16].

3.3.3 Mapping OCL Invariants

The OCL invariant specifies a condition on a class's attribute or association. Typically, an invariant is a predicate that should be TRUE in all possible worlds in UML class model's domain. The OCL context is specified in the invariants by using self keyword in place of the local variables.

3.3.4 Mapping OCL Pre/Post Conditions

Similar to the OCL invariant, the OCL preconditions and the OCL post condition are used specify conditions on operations of a class. Typically, a precondition is a predicate that should be TRUE before an operation starts its execution, while a post condition is a predicate that should be TRUE after an operation completes its execution [22].

3.3.5 Mapping OCL Expressions

The OCL expressions express basic operations that can be performed on available attributes of a class. An OCL expression in the OCL invariant can be used to represent arithmetic, and logical operations. OCL arithmetic expressions are based on arithmetic operators e.g. '+', '–', '/', etc., while, logical expressions use relational operators e.g. '<', '>', '=', '<>', etc. and logical operators e.g. 'AND', 'implies', etc.

3.3.6 Mapping OCL Operations

The OCL collections represent a set of attributes of a class. A number of operations can be performed on the OCL collections e.g. sum, size, forAll(), count, isEmpty, etc.

All the defined transformation rules defined in [23-25] were implemented in a java based library SiTra (Simple Transformation). The output of the OCL module is a complete OCL expression. The output OCL is saved and exported in a separate text file.

4. A CASE STUDY

In this section, we apply the qualitative evaluation criteria defined in Section 7.1and summarize the validation of our approach.

4.1 Experiment Details

This section discusses a used case study on the "Royal & Loyal" model. The Royal & Loyal model was originally presented for introducing OCL By Example in [26]. This case study is chosen as this case study is also done by Wahler [27] by his template based approach for automated generation of OCL constraints. The details of the case study are shown in Fig. 7 and Table 2.

4.2 Results of Evaluation

To evaluate the results of NL2OCLviaSBVR [16] tool, each result (OCL constraints) of the tool was compared with the sample results N_{sample} (opinion of the expert) and we have depicted opinion of the expert as OtherOCL. All the results were categorized as correct that was matched to the sample results and counted as correct result and denoted as $N_{correct}$. Similarly, the results of the tool not matched with the sample results were classified as incorrect results and denoted $N_{incorrect}$. In addition to this, the input that was missed by our tool and it was classified as the missing result and denoted by $N_{missing}$. Table 3 shows the results of the experiments:

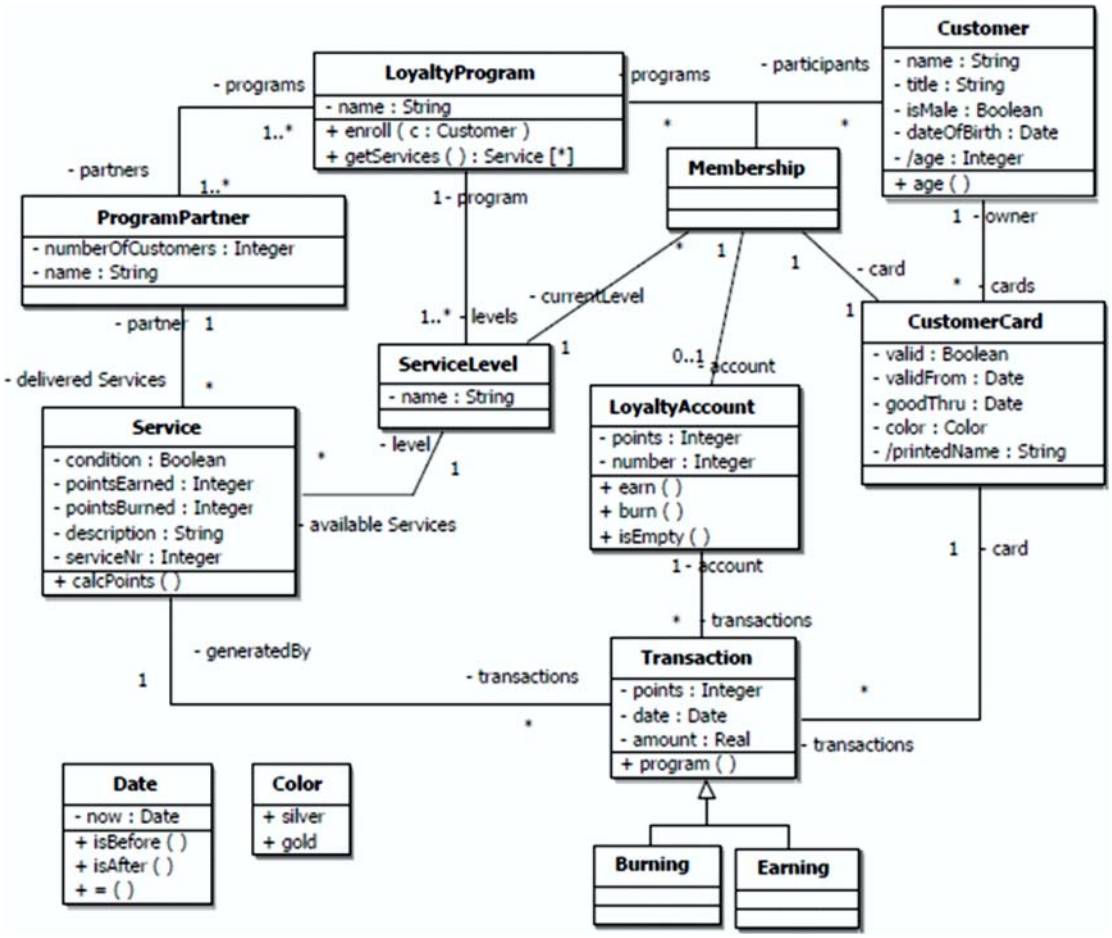
The transformation accuracy of each OCL constraint generated by our tool was tested by USE [28] by generating object diagrams of bot h OCL: OCL and Other OCL. If Object diagrams of both OCL are same the transformation is correct. Similarly, syntactic accuracy of each OCL constraint generated by our tool was verified by OClarity [29] tool. The achieved results were computed to overall recall 92.30% and precision is 96.15%. Such results are very much encouraging for future research and development a tool that can translate English constraints to formal representations.

A major focus of this paper is to compare the NL based approach (the NL2OCLviaSBVR [30] tool) for generating

OCL constraints with the template based approach (the Copacabana [27] tool). The results of the Royal and Loyal model were achieved by both types of the tools. The Copacabana tool can translate 18 English constraints to OCL that covers the 69.23% of the total specification. We have compared Recall value of Copacabana tool for the Royal & Loyal model constraints with NL2OCLviaSBVR in Table 4.

Here, it is obvious that the competency of the Copacabana tools is less than the NL2OCLviaSBVR tool. Moreover, the Copacabana tool is not fully automated. The user has to manually analyze the English software constraints, manually extract the information and then feed to the patterns that can generate OCL syntax. A comparison of the supported functionalities by the Copacabana tool and the NL2OCLviaSBVR tool is shown in Table 5.

It is shown in Table 5 that only NL2OCLviaSBVR supports all basic types of constraints such as invariants, pre-conditions and post-conditions. Moreover, The Copacabana tool does not support navigation via association classes. Moreover, the Copacabana tool does not support comparing the cardinality of sets. Here, the results shown in this paper are promising and envisages not only the framework discussed in this article but also the prospective of such work in general viewpoint.



FIG, 7. THE ROYAL & LOYAL MODEL

TABLE 2. SOLVED EXAMPLE OF ROYAL AND LOYAL MODEL

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 **customer card** must participate in at least one **loyalty program**.
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)

TABLE 3. EVALUATION RESULTS OF THE NL2OCLVIASBVR

Example	N_{sample}	$N_{correct}$	$N_{incorrect}$	$N_{missing}$	Rec%	Prec%	F-Value%
Results	26	24	1	1	92.30	96.15	94.12

TABLE 4. COMPARING NL2OCLVIASBVR TOOL AND COPACABANA TOOL

Tools for OCL Generation	Experiments' Recall (%)	Experiments' Precision (%)
NL2OCLviaSBVR [24]	92.30	96.15
Copacabana [21]	69.23	-

4.3 Limitations of the Tool

The designed system NL2OCLviaSBVR is always capable of producing the wrong analysis but that in such circumstances the produced formal representation is correct for a particular, valid and potentially correct interpretation and can be corrected by manual intervention. In particular, we have identified a few cases where the designed system has tendency to not generate the incorrect interpretation due to the following limitations.

- The NL2OCL approach works for a restricted domain. i.e., UML Class Model. Hence, the NL constraints should not contain the vocabulary outside the UML class model.
- The vocabulary names used in the NL constraints should be consistent with the vocabulary names used in the UML class model.
- NL constraints should be complete such as a NL constraint should have at least one valid context.
- Incomplete (if one side of the relation is missing) and invalid (wrong direction of the relation) relations such as associations, aggregations are not supported
- NL constraints should not have discrepancies neither among the used elements nor between the UML class models.
- NL constraints should not involve UML enumerations.
- NL constraint should not involve parameterized function calls.
- XOR relations in NL constraints are not supported.
- The OCL operationsoclTypeof(), oclIsKindOf(T), oclIsTypeOf(T), oclAsType(T), oclInState(s), sortBy(), count(), collect(), reject(), and append()are not supported in the tool.
- NL sentences should be declarative or imperative. The question based sentences are not processed.
- There are some limitations of the tool due to the use of the Stanford parser as a library. Major limitations of the Stanford parser in a role of NLP plugin are below:

TABLE 5. COMPARISON OF NL2OCLVIASBVR WITH OTHER TOOLS

Functionality	Copacabana	NL2OCL via SBVR
Invariants	Yes	Yes
Pre-conditions	No	Yes
Post-Condition	No	Yes
Collections	Semi	Semi
Logical Expressions	Yes	Yes
Relational Expressions	Yes	Yes
Conditional Expressions	Yes	Yes
Parameterized Function Calls.	No	No
Navigation	Semi	Yes
Queries	No	No

- A few times, the Stanford parser does not produce the right output after POS tagging due to lexical ambiguity in NL sentences. Since, the Stanford parser does handle lexical ambiguity by making a decision. However, this decision might be incorrect as it is not according to the interpretation the author intended.

- The NL2OCL approach is based on dependencies generated by the Stanford parser but the wrong typed dependencies are generated by the Stanford parser possibly due to semantic ambiguities. In such particular cases, due to wrong dependencies, wrong labeling of semantic roles can happen that result in irresolution of NL quantifiers, etc., in NL sentences.

- Since, the Stanford parser is not typically designed for the task we need, it does not generate correct output in case of some other ambiguities in NL sentences such as homonymy.

In this thesis, we have presented a novel approach to handle such ambiguities for which the Stanford parser does not produce the right output by using the information in the UML class model. However, there is a possibility that UML model does not contain the information to resolve a NL ambiguity and produce incorrect interpretation, and in such cases the user is involved to correct the output manually.

5. CONCLUSIONS

This research paper presents a framework for dynamic generation of the OCL constraints from the NL specification provided by the user. Here, the user is supposed to write simple and grammatically correct English. The designed system can find out the noun concepts, individual concepts, verbs and adjectives from the NL text and generate a structural or behavioral rule according to the nature of the input text. This extracted information is further incorporated to constitute a complete SBVR rule. The SBVR rules are finally translated to OCL expressions. SBVR to OCL translation involves the extraction of OCL syntax related information i.e. OCL context, OCL invariant, OCL collection, OCL types, etc. and then the extracted information is composed to generate a complete OCL constraint, or pre/post-condition.

As this paper aims to address a major challenge related to usability of OCL, we have presented a method of applying model transformations to create OCL statement from NL expressions. The presented transformation makes use of SBVR as an intermediate step to highlight the syntactic elements of NLs and make NL controlled and domain Specific. The use of automated model transformations ensures seamless creation of OCL statements and deemed to be non-intrusive. As a next step, we are hoping to investigate usability aspects of the tool directly via empirical methods involving teams of developers.

ACKNOWLEDGEMENTS

Authors acknowledge the support by OC Larity team and USE tool team for providing the assistance in using their tools for syntax accuracy and transformation evaluation. We are also thankful to the developers of Copacabana tool for providing the details for comparing the results of our tool Nl2OClviaSBVR.

REFERENCES

[1] OMG, "Unified Modeling Language (UML)", OMG Standard, Volume 2, No. 1, 2007.

[2] OMG, "Object Constraint Language (OCL)", OMG Standard, Volume 2, 2006.

[3] Wahler, M., "Patterns to Develop Consistent Design Constraints", Ph.D. Thesis, ETH Zurich, Switzerland, 2007.

[4] Gogolla, M., Büttner, F., and Richters, M., "USE: A UML-Based Specification Environment for Validating UML and OCL", Science of Computer Programming, Volume 69, No. 1, pp. 27-34, 2007.

[5] Cabot, J., "Ambiguity Issues in OCL Postconditions", Proceedings of 6th Conference OCL Workshop at the UML/MoDELS, pp. 194-204, 2006.

[6] Kristofer, J., "Disambiguation Implicit Constructions in OCL", Conference on OCL and Model Driven Engineering, Lisbon, Portugal, pp. 30-44, October 12, 2004.

[7] Correa A., Werner, C., and Barros, M., "An Empirical Study of the Impact of OCL Smells and Refactorings on the Understandability of OCL Specifications", MODELS, LNCS 4735, pp. 76-90, 2007.

[8] OMG, "Semantics of Business Vocabulary and Rules (SBVR)", OMG Standard, Volume 1. 2008.

[9] Linehan, M., "Ontologies and Rules in Business Models", 11th Conference Workshop in IEEE EDOC, pp. 149-156, 2008.

[10] Linehan, M., "SBVR Use Cases", Interntaional Symposium on Rule Representation, Interchange and Reasoning on the web, RuleML, LNCS, Volume 5321, pp. 182-196, 2008.

[11] OMG, "UML Superstructure Specification Document", OMG Standard, Volume 2, No. 3, 2007.

[12] Campbell, S., "Translation into the Second Language", Routledge, 2014.

[13] Harris, C.B., and Harris, I.G., "Generating Formal Hardware Verification Properties from Natural Language Documentation", IEEE International Conference on Semantic Computing, pp. 49-56, 2015.

[14] Whittle, J., and Jayaraman P., "MATA: A Unified Approach for Composing UML Aspect Models on Graph Transformation", Springer LNCS, Volume 5560, pp. 191-237, 2009.

[15] Kuhn, T., "A Survey and Classification of Controlled Natural Languages", Computational Linguistics, Volume 40, No. 1, pp. 121-170, 2014.

[16] Clark, T., Sammut, P., and Willans, J., "Applied Metamodelling: A Foundation for Language Driven Development", arXiv Preprint arXiv:1505.00149, 2015.

[17] Hirschberg, J., and Manning, C.D., "Advances in Natural Language Processing", Science, Volume 349, No. 6245, pp. 261-266, 2015.

[18] Soeken, M., Harris, C.B., Abdessaied, N., Harris, I.G., and Drechsler, R., "Automating the Translation of Assertions using Natural Language Processing Techniques", Forum on Specification & Design Languages, 2014.

[19] Gulwani, S., and Marron, M., "NLyze: Interactive Programming by Natural Language for Spreadsheet Data Analysis and Manipulation", Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 803-814, 2014.

[20] Bryant, B., "From Natural Language Requirements to Executable Models of Software Components", Workshop on SE for Embedded Systems, pp. 51, 2008.

[21] Akehurst, D.H., Boardbar, B., Evans, M., Howells, W.G.J., and McDonald-Maier, K.D., "SiTra: Simple Transformations in Java", ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, LNCS, Volume 4199, pp. 351-364, 2006.

[22] Cabot, J., Pau, R., and Raventós, R., "From UML/OCL to SBVR Specifications: A Challenging Transformation", Information Systems, Volume 35, No. 4, pp. 417-440, 2010.

[23] Bajwa, I.S., and Lee, M.G., "Transformation Rules for Translating Business Rules to OCL Constraints", 7th European Conference on Modelling Foundations and Applications, pp.158-163, Birmingham, UK, 2011.

[24] Bajwa, I.S., Bordbar, B., and Lee, M.G., "OCL Usability: A Major Challenge in Adopting UML", 2014 ICSE Workshop - RAISE, pp. 32-37, Hyderabad, India, 2014.

[25] Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, A., and Chanona-Hernández, L., "Syntactic n-Grams as Machine Learning Features for Natural Language Processing", Expert Systems with Applications, Volume 41, No. 3, pp. 853-860, 2014.

[26] Gogolla, M., Büttner, F., and Richters, M., "USE: A UML-Based Specification Environment for Validating UML and OCL", Science of Computer Programming, Volume 69, No. 1, pp. 27-34, 2007.

[27] Raj, A., Prabharkar, T., and Hendryx, S., "Transformation of SBVR Business Design to UML Models", ACM Conference on India Software Engineering, pp. 29-38, 2008.

[28] Demuth, B., and Wilke, C., "Model and Object Verification by Using Dresden OCL", RG Workshop on Innovation Information Technologies: Theory and Practice, pp. 81-89, 2009.

[29] IBM OCL Parser, http://www-01.ibm.com/ software/ awdtools/library/standards/ocl-download.htm, 2009.

[30] Burke, D., and Kristofer, J., "Translating Formal Software Specifications to Natural Language", Springer LNCS, Volume 3492, pp. 51-66, 2005.