Journal of King Saud University – Computer and Information Sciences (2012) **24**, 117–128

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

Translating natural language constraints to OCL

Imran Sarwar Bajwa *, Mark Lee, Behzad Bordbar

School of Computer Science, University of Birmingham, B15 2TT, United Kingdom

Received 8 November 2011; accepted 31 December 2011
Available online 10 January 2012

KEYWORDS

Natural language processing;
English constraints;
Formal constraints

Abstract Object Constraint Language (OCL) is the only available language to annotate the Unified Modeling Language (UML) based conceptual schema (CS) of a software application. In practice, the constraints are captured in a natural language (NL) such as English and then an OCL expert manually transforms the NL expressions to OCL syntax. However, it is a common knowledge that OCL is difficult to write specifically for the novel users with little or no prior knowledge of OCL. In recent times, model transformation technology has made transformation of one language to another simpler and easier. In this paper we present a novel approach to automatically transform NL specification of software constraints to OCL constraints. In NL to OCL transformation, Semantics of Business Vocabulary and Rules (SBVR) standard is used as an intermediate representation due to a couple of reasons: first of all, SBVR is based on higher order logic that simplifies the transformation of SBVR to other formal languages such as OCL. Moreover, SBVR used syntax of natural language and thus is close to human beings. The presented NL to OCL transformation via SBVR will not only simplify the process of generating OCL constraints but also generate accurate models in less time.

1. Introduction

In software engineering, people need to translate a piece code from one language to another to make reuse of the existing code. Similarly, the natural language (NL) expressions are translated to formal languages (such as Java, C#) to makes them machine-processable. However, the manual translation cannot only be time consuming but also error-prone. To facilitate the process of language translation, automatic translations are proposed in computer science, e.g. NL to UML (Unified Modeling Language) class models (Harmain and Gaizauskas, 2003), NL software requirements to Java (Price et al., 2000), and NL queries to SQL (Structured Query Language) queries (Giordani, 2008). However, a key issue in automatic translation of natural language expressions to formal language expressions is low accuracy of translation that is reported nearly 65–70% in real time software development. It has also been identified that a primary reason of less accuracy is the ambiguous nature of NL. Kiyavitskaya et al. (2008) identified that various lexical, syntactic and semantic ambiguities in natural languages make machine processing of NL not only complex but also erroneous.

In modern software engineering, the graphical models such as UML class models are commonly used to represent a conceptual schema (CS) of a software application. But the UML

* Corresponding author.
E-mail addresses: i.s.bajwa@cs.bham.ac.uk (I.S. Bajwa), m.g.lee@cs.bham.ac.uk (M. Lee), b.bordbar@cs.bham.ac.uk (B. Bordbar).

Peer review under responsibility of King Saud University.

ELSEVIER Production and hosting by Elsevier

1319-1578 © 2012 King Saud University. Production and hosting by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jksuci.2011.12.003

models are incomplete without textual constraints. The textual such as OCL (OMG, 2010) constraint is specifically used to improve precision of UML class models. However, various researchers have proved that OCL syntax is difficult, especially for novel users (Cabot, 2006; Wahler, 2008). Moreover, it has also been discovered that various complexities in OCL syntax make it difficult to write OCL (Cabot and Teniente, 2007) and a manual effort in writing the OCL code may result in erroneous and inconsistent OCL expressions (Wahler, 2008). Therefore, it is commonly attributed that OCLs difficult syntax is the main hindrance in OCL wide adaptation and acceptance in the software modeling community despite the fact that OCL is an integral part of UML based software models.

In this paper, we have addressed this challenging question by proposing a natural language based approach to automatically generate OCL code. A key challenge in translation of English specification of software constraints to OCL was tackling of the inherent syntactic and semantic ambiguities in English. In the presented approach, the input English specification of software constraint is syntactically and semantically analyzed to generate a SBVR based logical representation. A SBVR based logical 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). Finally, the SBVR representation is mapped to OCL using model transformation technology. The presented approach will boost the acceptance of OCL in software designers and developers community.

The remaining paper is structured into the following sections: Section 2 provides an overview of SBVR, OCL and model transformation technology. Section 3 illustrates the architecture of NL*2*OCL approach. Section 4 presents a case study. The evaluation of our approach is presented in Section 5. Finally, the paper is concluded to discuss future work.

2. Preliminaries

2.1. Semantics of Business Vocabulary and Rules

In 2008, a new standard Semantics of Business Vocabulary and Rules (SBVR) (OMG, 2008) was introduced by OMG to support business and software community. A typical SBVR representation is based on a set of business vocabulary and business rules. Fig. 1 shows an overview of the subset SBVR (meaning) metamodel.

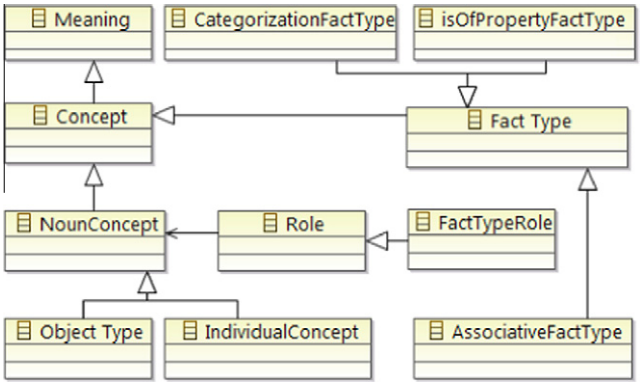

Figure 1 A subset of SBVR metamodel.

SBVR business vocabulary. A SBVR business vocabulary (OMG, 2008, section: 8.1) consists of all the specific terms and definitions of concepts used by an organization or community in the course of business. In SBVR, a concept can be a noun concept or fact type. A noun concept can be an *Object Type*, or *Individual Concept*, *Verb Concept* or a *Characteristic*.

SBVR business rule. A SBVR business rule consists of all the specific terms and definitions of concepts used by an organization or community in the course of business. In SBVR, a concept can be a noun concept or fact type. A noun concept (OMG, 2008, section: 8.3) can be an *Object Type*, or *Individual Concept*, *Verb Concept* or a *Characteristic*.

SBVR semantic formulation. SBVR proposes a set of semantic formulations that (OMG, 2008, section: 8.4) semantically formulate the SBVR rules. In SBVR 1.0, there are five types of semantic formulations. However, we have used following three of those that relate to the scope of our research:

- Logical operations: Various number of logical operations such as conjunction, disjunction, implication, negation, are supported.
- Quantification: A set of quantifications supported in SBVR consists of universal quantification, at least *n* quantification, at most *n* quantification, etc.
- Modal formulation: There are set of modal formulations also available in SBVR, e.g. "It is obligatory" or "It is necessary". The modal formulations are used to formulate modality.

2.2. Object Constraint Language

Object Constraint Language (OCL) (OMG, 2007) is the only available language used to annotate the UML class model with the constraints. OCL supports three types of constraints: invariants, pre-condition and post-condition. OCL can also be used for representing queries but that is out of the scope of the research.

Typically, a constraint is a restriction on state or behavior of an entity in a UML model. The OCL constraint defines a Boolean expression. If the constraint results are true, the system is in valid state. In this paper, we target the generation of OCL invariants. The invariants (OMG, 2007) are conditions that have to be TRUE for each instance of the model.

3. Translating English to logical representation

In our research, we aim to generate OCL constraints from natural language (NL) specification using SBVR as a pivot representation. The first step in NL to OCL translation was the understanding of NL specification for the generation of a logical representation that can be mapped to SBVR and finally to OCL constraints. This section briefly explains how the English text is mapped to a SBVR based logical representation. The approach works in the following four phases.

3.1. Lexical analysis

In the preprocessing phase, the input text containing the natural language specification of an OCL constraint for a UML

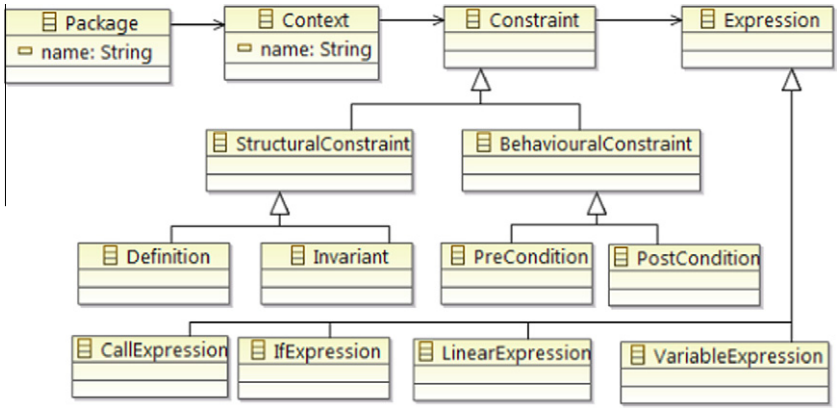

Figure 2 Elements of selected OCL meta-model.

class model is preprocessed for deep processing. Major steps involved in preprocessing phase are splitting the sentences, tokenization, and lemmatization. Following is a brief discussion of all the preprocessing sub-phases:

Sentence splitting. In the 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. Examples of tokenized text are shown in Figs. 2 and 3.

POS tagging. The tokenized text is processed by the Stanford POS (part-of-speech) (Toutanova et al., 2003) tagger to identify part of speech for each token in the input text. The Stanford POS tagger version 3.0.3 has been used to identify 44 various parts of speech. Typically, accuracy of the Stanford POS tagger is very high up to 97% (Manning, 2011). However, in a few cases, the Stanford POS tagger is not able to identify the correct POS tags (Bajwa et al., 2012). An example of such cases is shown in Fig. 4 where, the token "books" is identified as a noun. However, the token "books" is a verb and the correct POS tag is "VBZ". As, we are using the Stanford parser for generating parse tree and the Stanford parser uses output of the Stanford POS tagger, this problem becomes more serious as wrong POS tags lead to a wrong parse tree.

We have addressed cases by mapping all the tokens to the target UML class model. If a token matches to an operation-name or a relationship name then it is a verb or if a token matches to a class-name or attribute-name then it is classified as a noun.

Lemmatization. In lemmatization, the morphological analysis of words is performed to remove the inflectional endings and to return to the base or dictionary form of a word, which is known as the *lemma*. We identify lemma (base form) in the POS tagged tokens (all nouns and verbs) by removing various suffixes attached to the nouns and verbs, e.g. in Fig. 3, a verb "awarded" is analyzed as "award + ed". Similarly, the noun "workers" is analyzed as "worker + s" (see Fig. 5).

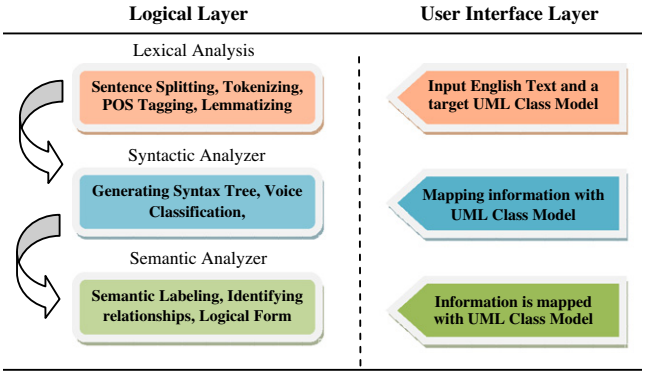

Figure 3 Processing natural language software constraints.

3.2. Syntactic analysis

The output of a typical syntax analysis phase is a parse tree diagram or other textual representation. Our syntactic analyzer uses the Stanford parser to parse the preprocessed text by. In syntax analysis phase, following three steps are performed:

Generating syntax tree. We have used the Stanford parser to generate parse tree from lexically analyzed text. The Stanford parser is a lexically driven probabilistic parser. The Stanford parser is a Java implementation of a probabilistic natural language parser based on Probabilistic Context-Free Grammars (PCFG). The Stanford parser provides to outputs: a phrase structure tree and a Stanford dependencies output. The typed dependencies generated by the Stanford parser represent the grammatical relations in an English sentence (Marneffe et al., 2006). The Stanford parser provides 84.1% accuracy

English:	A student books two novels.
Tokens:	[A/DT] [student/NN] [books/NNS] [two/CD] [novels/NNS] [./.]

Figure 4 Parts-of-speech tagged text.

English: A burger can be served to any customer with fries.

Parse Tree:

```
(ROOT
  (S
    (NP (DT A) (NN burger))
    (VP (MD can)
      (VP (VB be)
        (VP (VBN served)
          (PP (TO to)
            (NP
              (NP (DT any) (NN customer))
                (PP (IN with)
                  (NP (NNS fries))))))))
    (. .)))
```

Typed dependencies, collapsed:

```
det(burger-2, A-1)
nsubjpass(served-5, burger-2)
aux(served-5, can-3)
auxpass(served-5, be-4)
root(ROOT-0, served-5)
det(customer-8, any-7)
prep_to(served-5, customer-8)
prep_with(customer-8, fries-10)
```

Figure 5 Syntactic tree generated using Stanford Parser.

(Cer et al., 2010). However, we have experienced that the Stanford parser generates wrong typed dependencies for some English sentences. An example of such sentences is shown in Fig. 5, where token "customer" is wrongly related with the token "fries". Whereas, that correct typed dependency should be prep_with (customer-8, fries-10) to represent the actual meanings of the example i.e. the customers are served with burger and fries.

To handle such inaccurate syntactic analysis, we have used the information available in the UML class model. Fig. 6 shows a UML class model that can help us to identify the correct dependencies of the example shown in Fig. 5. The relationships in the UML class model such as associations (directed and un-directed), aggregations and generalizations can help us to deal with such ambiguities of English. By mapping the English text information with the UML class model, such ambiguities can be addressed as the UML enumeration class, i.e. for example the food class hints that burger is related with fries.

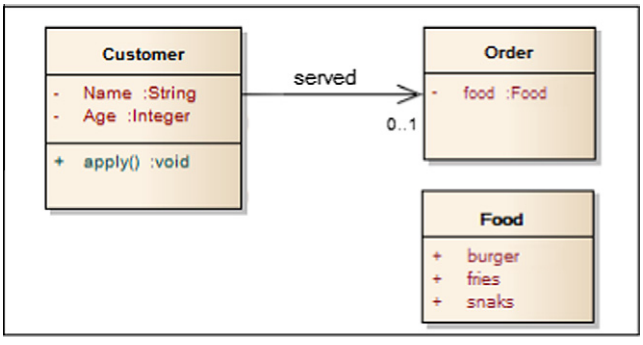

Figure 6 A UML class model.

Voice classification. In the voice classification phase, the sentence is classified into active- or passive-voice categories. Typically, the passive voice implies focus on the grammatical patient in place of the agent of the sentences. Various grammatical features manifest passive-voice representation such as the use of past participle tense with main verbs and can be used for the identification of a passive-voice sentence (see Fig. 7). Similarly, the use of the "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. By using this information, a set of rules was defined to classify the voice of a sentence.

After voice classification, various parts of a sentence are classified into a subject, verb or object. In case of a NP relation, there can be more than one subject or object relating to a verb. Similarly, in a VP relation, more than one verb can relate to a subject. This process is also called shallow syntactic parsing in which a sentence is analyzed to identify various constituents such as subject, verb, object, etc.

Processing conjunction and disjunction. Logical operators are significant parts of many English sentences. Logical operators such as conjunction and disjunction can be analyzed by syntactic information. In Syntax analysis, we have also identified conjunctions and disjunctions. Following sub-sections explain the way conjunctions and disjunctions are handled:

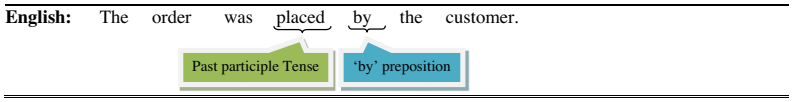

Figure 7 Identifying passive-voice sentences.

i. *Conjunction*. We have used the parse tree information to identify conjunction ($p \wedge q$) in English sentences. Typically, conjunction is represented using a few words such as "*and*", "*but*", "*yet*", "*so*" "*moreover*", "*however*", "*although*", "*even* though", etc. Conjunction can be fused to join two nouns or two verbs. The "and" conjunction used with two nouns is easy to interpret e.g. "A student and teacher can borrow a book". However, use of "*and*" conjunction with two verbs can be ambiguous e.g. "John opened the door and went out". In this example, and is not serving as a conjunction but an implication. However, we have not currently handled implicatures.

ii. *Disjunction*. In natural language text, disjunction can be inclusive or exclusive. Typically, inclusive disjunction ($p \vee q$) means either p is true or q is true or both. In English, inclusive disjunction represented using "or" word. Similar to "and", the use of "or" is also ambiguous in English as sometimes it disjoins nouns/adjectives and some times disjoins two propositions. We have identified this difference and defined simple rules to classify the different use of "or" in all three possible situations. For example "A student can borrow a book or a CD." Other possible representation of inclusive disjunctions in English can be use of "unless", "and/or".

iii. *Exclusive disjunction*. ($p \oplus q$) (XOR) is also used in English e.g. "Do you want milk or sugar in your coffee?" does not gives a constraint "milk" XOR "sugar". In another context the example, "do you want milk and sugar in your coffee?" suggests "milk" AND "sugar". Both are examples of inclusive OR. We are aware of such subtle aspects. However, we have not handled this type of relation until mentioned explicitly.

Generating an intermediate representation. In this phase, an intermediate representation is generated for the further semantic analysis in the next phase. A tabular representation is generated containing the syntactic chunks and their associated representation such as syntax type (such as subject, verb or object), quantification, logical operator, and associated preposition (Table 1).

A major feature of this intermediary representation is that the active voice and passive voice are mapped to the 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.

3.3. Semantic analyzer

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 (Lu et al., 2008). The goal of semantic analysis is to understand the exact meanings of the input text and identify that relationship in various chunks. For a complete semantic analysis of domain specific text, we have to analyze the text in respect of particular domain such as the UML class model. Domain specific text analysis demands knowledge from the application domain to be mapped with the input English. In our research, UML class model is an application domain of the input NL specification of constraints. Our semantic analyzer performs the following three steps to identify relations in various syntactic structures:

1. Shallow semantic parsing.
2. Intermediary semantic representation.
3. Deep semantic parsing.

Shallow semantic parsing. In shallow semantic parsing the semantic or thematic roles are typically assigned to syntactic structure in a NL sentence. This process is also called Semantic Role Labeling. The actual purpose of semantic role labeling is identifying relationship of participants (semantic arguments) with the main verb (semantic predicate) in a clause. SRL is a most common way of representing lexical semantics of NL text. 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:

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

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,

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

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:

a. Object Type → Common nouns,
b. Individual concept→ Proper nouns,
c. Verb concepts → Main verb and
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:

i. Identifying predicates. In the first step, the system identifies the words in the sentence that can be semantic predicates or semantic arguments, and for which semantic roles need to be found and annotated. For identifying predicates we need following information that we extracted in a previous (syntactic analysis) phase for the input sentence "Customer places order".

Table 1 An intermediary logical representation of an NL sentence.

#	Chunk	Syntax	Quantification	Logical operator	Preposition	EOS
1	Customer	Subject	1			
2	Can	H.Verb		Not		
3	Place	M.Verb				
4	Order	Object	More than 1			True

Above shown information is extracted by using lexical and structural features identified. Such typical features assist in manifesting a semantic predicate, a semantic argument and relations between predicate and arguments. By using this syntactic information, we have identified predicates in the following two phases:

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 (see Fig. 9(a)). 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 (see Fig. 8).

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

Extracting semantic arguments. By excluding the pre-modifiers and post modifiers, we can extract the noun concepts. For further classification of object types and individual concepts the POS type of the noun concept is checked. If the POS type is a common noun, it is categorized as an *Object Type* and if the POS type is proper noun, it is categorized as an *individual concept*. Once the noun concepts are extracted, the next phase is to process phrases to generate a semantic representation. We have identified three types of phrases in typical constraints as following (see Figs. 9(b) and 10):

a. *Processing phrases*. Typical phrases are a combination of two or more than two words. In SBVR, both the object types and individual concepts are represented in the form of phrases. Following two examples show how phrases are processed to a semantic representation:
English: credit customer,
FOL: is a (x, customer) ∧ object_type (x, credit),
English: gold credit customer,
FOL: is a (x, customer) ∧ object_type (x, credit) ∧ object_type (x, gold).

b. *Generative noun phrase*. The generative noun phrases are also very common in constraints. Especially the SBVR characteristics are described by using generative noun phrases e.g. customer's age, customer's salary, etc. Following examples show the way we have processed generative noun phrases to a semantic representation.
English: customer's account.
FOL: object_type (x, customer) ∧ characteristic (x, account).
English: account of customer.
FOL: object_type (x, customer) ∧ characteristic (x, account).

c. *Adjective phrases*. Adjective phrases are not common in constraints but we have processed the adjective phrases as they can be a possible case. Following are the examples showing the processing of adjective phrases:
English: The customer is happy.
FOL: is (x, customer) ∧ characteristic (x, happy).
English: This is a gold customer.
FOL: is a (x, customer) ∧ characteristic (x, gold).

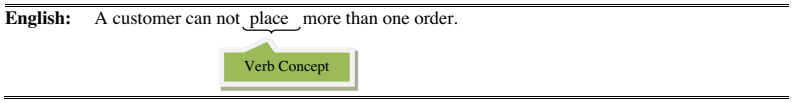

Figure 8 Identifying verb concepts (predicate).

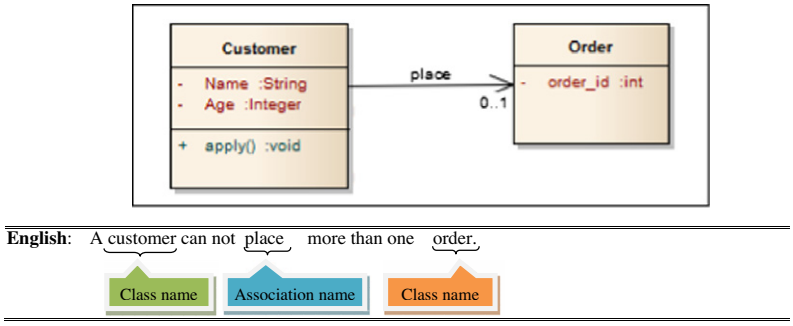

Figure 9 (a) UML class model and (b) English sentence mapped with class model.

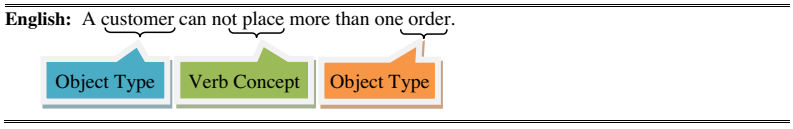

Figure 10 Semantic roles assigned to input English sentence.

ii. Sense recognition After identification of predicate, we need to recognize the exact sense of the predicates so that accurate semantic roles may be assigned to the predicate. Sense of a predicate is identified according to the target UML model. For example a verb can be an operation of a class or also can be an association of two classes. Another purpose of this mapping is the confirm that each element of the input is also part of the target UML model and such mapping will ensure that the finally generated representation will be consistent to the target UML class model.

iii. Thematic role classification After sense recognition, we have to decide the exact semantic label/role for a particular substring. The substrings are labeled with a semantic role. The used approach for semantic role labeling works as the syntactic tree representation of a sentence is mapped into a set of syntactic constituents. Finally, each syntactic constituent is classified into one of semantic roles. The classification is performed on the basis of the sentence structural features or the linguistic context of the target constituent. Semantic role classification is performed as the syntactic information (such as part of Speech and syntactic dependencies) is used to identify predicates and predicate roles. Output of semantic role labeling phase is semantic predicates and semantic arguments labelled with their corresponding roles.

Deep semantic analysis. Typically computational semantics aim at grasping the entire meanings of the natural language sentence, rather than focusing on text portions only. For computational semantics, we need to analyze the deep semantics of the input 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 a most common aspect quantification resolution:

i. *Resolving quantifications*. In natural languages, quantifications are typically expressed with noun phrases (NPs). However, in First-Order Logic (FOL), the variables are quantified at the start of the logical expressions. Generally, the natural language quantifiers are much more vague and varied. This vagueness makes translation of NL to FOL complex. However, we have done following two things to handle quantifiers variable scoping:

With respect to our target representation (SBVR rules), we have identified following four types of quantifications that we need to handle as SBVR 1.0 also support these four types of quantifications. First two quantifications such as universal and existential are most common. However, these two types do not cover all possible types in detail. We cover all possible types of quantifications in natural languages; we have used two other types such as uniqueness and solution quantification. Hence, it will be simple to map these NL quantifications to SBVR quantifications.

Universal quantification ($\forall X$). The universal quantifier is represented using *all* sign "$\forall$" and means all the objects X in the universe. The universal quantification is mapped to *Universal Quantification* in SBVR. The NL quantification structures "each", "all", and "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 due to the fact that we are processing constraints and generally constraints are mentioned for all the possible X in a universe (see Fig. 11).

Existential quantification ($\exists X$). The existential quantifier is represented using *exists* sign "$\exists$" and means at least one object X exists in the universe. 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.

Uniqueness quantification ($\exists_{=1} X$). The uniqueness quantifier is represented using "$\exists_{=1}$" or "$\exists_!$" means exactly one object X in the universe. The uniqueness quantification is mapped to *Exactly-One* Quantification in SBVR. The determiners "a" and "*n*" used with object part of the sentence are treated as uniqueness quantification.

Solution quantification (§X). The solution quantifier (Hehner, 2004) is represented using section "§" sign and means *n* object in the universe. 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* (see Fig. 8). Similarly, if the keywords like less than or smaller than are used with *n* then solution quantifier is mapped to *At-least Quantification*.

Two other types of quantifications are also available such as Plaucal quantification ($\exists^{many} X$) and multal Quantification ($\exists^{few} X$). However, we are not using these both quantifications as both of them are not supported by SBVR and UML and ultimately cannot be translated to OCL.

i. *Quantifier scope resolution* In quantification resolution, second issue is quantifier scope resolution. For quantification variable scoping, we have treated syntactic structures as logical entities

ii. *Resolving logical operations*.

Negation. Negation is an important construct that is used to negate a structure by using keywords *no* and *not* e.g. "A

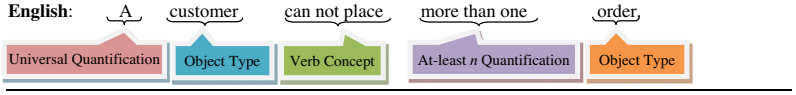

Figure 11 Semantic roles assigned to input English sentence.

English: A customer can place one order.

Semantic Interpretation:

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

Figure 12 Semantic roles assigned to input English sentence.

customer cannot apply for more than one account." Here, negation has been used to restrict customers to a single account. We have also worked out the double negation as a positive sentence. Hence, $\neg(\neg p) = p$.

Implication. In English, a few expression are used to represent implications such as "if p, then q", "if p, q", "q if p", "p only if q", "p implies q", "p entails q", "p hence q", "q provided p", "q follows from p". For example "If student is adult, he can get a pass". We have also identified that some expressions such as "q since p", "since p, q", "because p, q", "q because p", "p therefore q" are not true cases of implications.

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. Fig. 12 shows an example of the semantic interpretation we have used in the NL to OCL approach:

4. Translating logical form to OCL

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

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

4.1. 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 [UML-Class]
inv: [Body]
```

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

```
package [UML-Package]
context [UML-Class::Class-Op(Param):Return-Type]
pre: [Body]
```

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

```
package [UML-Package]
context [UML-Class::Class-Op(Param):Return-Type]
post: [Body]
Result: [Body] – optional
```

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

i. *UML-Package* is the package name of the target UML class model.
ii. *UML-Class* is the name of the class in the target *UML Class* model and UML-Class should also be an Object Type in the subject part of the English Constraint.
iii. *Class-Op* is one of the operations of the target class (such as context) in the UML class model and *Class-Op* should also be the Verb Concept in English constraint.
iv. *Param* is the list of input parameters of the *Class-Op* and we get them from the UML class model. These parameters should be of type Characteristics in English constraint.
v. *Return-Type* is the return data type of the *Class-Op* 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.

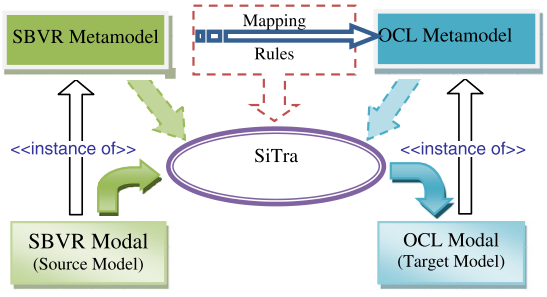

Figure 13 SBVR to OCL transformation framework.

vi. *Body* can be a single expression or combination of more than one expression. The details of *Body* are given in the next section.

4.2. Mapping logical form to OCL

The *Body* of the invariants and the pre/post-conditions is generated from the logical form generated in Section 2.2. A set of transformation rules (Bajwa and Lee, 2011) were used to translate SBVR based logical representation to OCL by mapping element(s) of the SBVR metamodel to equivalent element(s) of the OCL metamodel (Bajwa et al., 2010). Sitra library was used in this mapping as shown in Fig. 13.

Moreover a set of mappings were used to map logical elements to OCL elements. Following is a brief overview of the used mappings from logical representation to OCL:

Logical expression: In OCL, two expressions are concatenated using a logical operator. Following are the possible cases of logical expressions:

English representation	OCL rep.
p or q	`p or q`
p and q, p but q, p yet q, p so q, p moreover q, p however q, p although q, p even though q	`p and q`
p then q, if p q, q if p, p only if q, p implies q, p entails q, p hence q, q provided p, q follows from p	`p implies q`

Relational expressions: In OCL, two expressions can be concatenated using a relational operator. Following are the possible cases of relational expressions:

English representation	OCL rep.
p is q, p is equal to q	`p=q`
p is greater than q, p is larger than q, p is more than q,	`p>q`
p is less than q, p is smaller than q	`p<q`
p is more than or equal to q	`p>=q`
p is less than or equal to q	`p<=q`

Postfix expressions: In OCL, there can be a postfix expression such as `self`.

English representation	OCL rep.
attribute of the context class	`self.`[*attribute*]

Navigation: The navigation expressions are most common expressions in OCL.

English representation	OCL rep.
p's q, q of P	`p.q`
p is q()	`p.q()`
size of p, number of p	`p -> size()`
number of p in q	`p -> count(q)`
p is empty	`p -> isEmpty()`
sort p	`p -> sortBy()`
q exists in p	`p -> exists(q)`

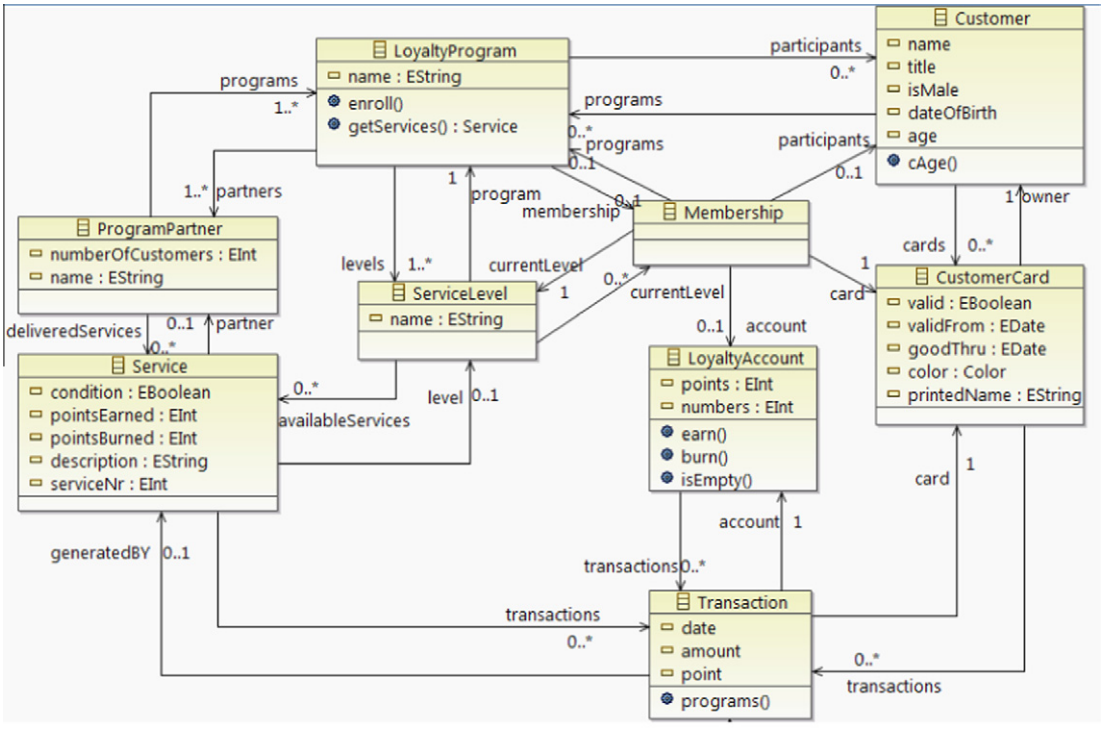

Figure 14 The Royal & Loyal model.

Conditional expression: In OCL, there can be a conditional expression. The conditional expressions can be of two types: if–then expressions, and if–then–else expressions.

English representation	OCL rep.
if p then q	`if` [*Relational-Exp*] `then` *q* `endif`
if p then q else R	`if` [*Relational-Exp*] `then` *q* `else` *R* `endif`

5. Case study

A case study the "Royal & Loyal" model was solved using our tool NL2OCL*via*SBVR to test the accuracy of the presented approach. The Royal & Loyal model was originally introduced in *OCL By Example* in Kleppe and Warmer (2003). Afterward, the Royal & Loyal model is used in various publications, e.g., Tedjasukmana (2006), Dzidek et al. (2005) and Wahler (2008). The same model is also shipped with several tools as an example model, e.g., Dresden Technical University (2007).

5.1. The Royal & Loyal model

The Royal & Loyal model is a computer system of a company that handles loyalty programs for its various customers. Key participating classes are LoyaltyProgram (see Fig. 14), `Customer` and class `ProgramPartner` and `LoyaltyProgram`. Other participant classes are `Membership`, `LoyaltyAccount`, `CustomerCard` and `Transactions`. In the Royal & Loyal model, the program partners various services and each membership is associated with exactly one service level.

5.2. The Royal & Loyal model constraints

Wahler has discussed 26 English constraints of the Royal & Loyal case study in his PhD thesis. However, his (pattern-based) approach is able to translate just 18 English constraints into OCL that is 69% of the total constraints. In comparison to Wahler's pattern based approach, our NL-based approach has successfully translated all 26 constraints to OCL. 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. In the following section, we present just three out of 26 constrains due to shortage of space.

Constraint 1

English: There must be one transaction with exactly 500 points.
SBVR: It is necessary that there *must be* one transaction with exactly 500 "point".
OCL:
```
package: royal_and_loyal
    context Transaction
    inv self.transaction->select(point = 500)->Size()=1
```

Constraint 2

English: All cards that generate transactions on the loyalty account must have the same owner.
SBVR: It is necessary that each all *card* that *generates* transactions on the loyalty account *must have* the same *owner*.
OCL:
```
package: royal_and_loyal
    context Loyalty Account
    inv self.transactions.cards.owner->asSet()->size() = 1
```

Constraint 3

English: A maximum of 10,000 points may be earned using services of one partner.
SBVR: It is a possibility that a maximum of 10,000 *points may be* earned using the services of one partner.
OCL:
```
package: royal_and_loyal
    context ProgramPartner
    inv self.delivered Services.points earned <= 10,000
```

Constraint 4

English: The maximum age of participants in loyalty programs is 70.
SBVR: It is necessary that the at most "age" of participants in loyalty program *is* 70.
OCL:
```
package: royal_and_loyal
    context LoyaltyProgram
    inv self.participants->for all(ages <= 70)
```

6. Evaluation

To test the accuracy of the OCL constraints generated by the designed system two classes were defined: preconditions and post-conditions. Invariant class was not used as it is irrelative to the used case study. The package merges English constraints and transformations were classified into three classes with respect to complexity levels of input, i.e. simple, compound and complex.

6.1. Evaluation methodology

For evaluation of the designed system, a criterion was defined that how close are the constraints generated by our system (named *system results*) to the constraints produced by the human experts (named *sample results*). As, there can be multiple representations for a single constraint, different human experts produce different representations that can be good or bad constraints. However, we gained a human expert's constraints for the target input and used it as a sample result. We have used three evaluation metrics [34], [35] for NL2OCL*via*SBVR: recall, precision and *F*-value. These metrics are extensively employed to evaluate NL-based knowledge extraction systems.

Table 2 Evaluation results of NL2OCL*via*SBVR.

Input	N_{sample}	$N_{correct}$	$N_{incorrect}$	$N_{missing}$	Rec%	Prec%	*F*-value
C1	48	37	8	3	77.08	82.22	79.65
C2	43	33	8	2	76.74	80.48	78.61
C3	39	31	5	3	79.48	86.11	82.79
C4	36	29	3	4	80.55	90.62	85.58
C5	26	24	1	1	92.30	96.15	94.12
Average					81.23	87.12	84.15

Table 3 Usability survey results.

User	Easy to use		Time-saving		Correctness	
	Manual (%)	By tool (%)	Manual (%)	By tool (%)	Manual (%)	By tool (%)
Novel	30	90	25	85	15	65
Medium	55	85	40	80	50	70
Expert	70	85	60	70	80	80
Average	51.66	86.66	41.66	78.33	48.33	71.66

6.2. Evaluation results

Four other case studies were solved in addition to the case study presented in Section 5. All the case studies were unseen. The solved case studies were of different lengths. Calculated recall, precision and *F*-values of the solved case studies are shown in Table 2.

The average *F*-value is calculated 84.15% that is encouraging for initial experiments. We cannot compare our results to any other tool as no other tool is available that can generate OCL constraints from NL specification. However, we can note that other language processing technologies, such as information extraction systems, and machine translation systems, have found commercial applications with precision and recall figure well below this level. Thus, the results of this initial performance evaluation are very encouraging and support both NL2OCL approach and the potential of this technology in general.

6.3. Usability survey

A small survey was conducted to measure the effectiveness of the presented approach. For the survey three groups were defined as below:

Novel : A user who is quite new to OCL.
Medium: A user who knows basics of OCL.
Expert: A user who is an expert of OCL.

Each group consists of 10 users. A set of inputs such as English specification of OCL constraints were provided to all the users. First all the users were asked to solve the input manually and then they were asked to generate the OCL constraints by using our tool NL2OCL*via*SBVR. Once all the users finished their work they were given a questionnaire to fill. In the questionnaire, questions were asked regarding various aspects: simple to use, time-saving, correctness, etc. Each user was asked to give 1–10 scores for each category. The average results we received are shown in Table 3.

The average values calculated for different parameters are clearly showing that the used approach was clearly making an impact. Though the accuracy of the tool is a bit of concern we can overcome this in future work by improving the implementation.

7. Conclusion

The current presented work focuses on automated (object oriented) analysis of NL specification and the 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 the benefit of our tool as the NL2OCL can generate accurate OCL constraints with less effort. However, our tool is limited to process one English sentence at a moment. In future, we aim to enhance our tool to process multiple constraints.

References

Bajwa, I.S., Lee, M., 2011. Transformation rules for translating business rules to OCL constraints. In: 7th European Conference on Modelling Foundations and Applications (ECMFA 2011), Birmingham, UK, June, pp. 132–143.

Bajwa, I.S., Bodbar, B., Lee, M., 2010. OCL constraints generation from natural language specification. In: 14th IEEE International Enterprise Distributed Object Conference (EDOC 2010), Vitoria, Brazil, October, pp. 204–213.

Bajwa, I.S., Bodbar, B., Lee, M., 2012. Resolving syntactic ambiguities in natural language specification of constraints. In: 13th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2012), Delhi, India.

Cabot, J., 2006. Ambiguity issues in OCL post-conditions. In: Proc. OCL for (Meta-) Models in Multiple Application Domain – MODELS'06, Technical Report.

Cabot, J., Teniente, E., 2007. Transformation techniques for OCL constraints. Journal of Science of Computer Programming 68 (03), 152–168.

Cer, D., Marneffe, M.C., Jurafsky, D., Manning, C.D., 2010. Parsing to stanford dependencies: trade-offs between speed and accuracy. In: Proceedings of LREC-10.

Dzidek, W., Briand, L., Labiche, Y., 2005. Lessons learned from developing a dynamic OCL constraint enforcement tool for Java. LNCS 3844, 10–19.

Giordani, A., 2008. Mapping natural language into SQL in a NLIDB. Natural Language and Information Systems 5039, 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.

Hehner, E.C.R., 2004. Practical Theory of Programming, second ed. p. 28.

Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D., 2008. Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements Engineering 13 (3), 207–239.

Kleppe, A., Warmer, J., 2003. The Object Constraint Language, second ed. Addison-Wesley.

Lu, W., Ng, H.T., Lee, W.S., Zettlemoyer, L.S., 2008. A generative model for parsing natural language to meaning representations. In: Empirical Methods in Natural Language Processing (EMNLP), vol. 55, pp. 55–56.

Manning, C.D., 2011. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In: Proceedings of CICLing, vol. 1, pp. 171–189.

Marneffe, M.C., MacCartney, Bill, Manning, C.D., 2006. Generating typed dependency parses from phrase structure parses. In: LREC, vol. 55, pp. 55–56.

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., Harvey, B., 2000. NaturalJava: a natural language interface for programming in Java. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI).

Tedjasukmana, V.N., 2006. Translation of OCL invariants into SQL: 99 integrity constraints. Master's Thesis, Technical University of Hamburg, Germany.

Toutanova, K., Klein, D., Manning, C., Singer, Y., 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of HLT-NAACL, pp. 252–259.

Wahler, M., 2008. Using patterns to develop consistent design constraints. Ph.D. Thesis, ETH Zurich, Switzerland.