2010 14th IEEE International Enterprise Distributed Object Computing Conference

OCL Constraints Generation from Natural Language Specification

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

Abstract— **Object Constraint Language (OCL) plays a key role in Unified Modeling Language (UML). In the UML standards, OCL is used for expressing constraints such as well-definedness criteria. In addition OCL can be used for specifying constraints on the models and pre/post conditions on operations, improving the precision of the specification. As a result, OCL has received considerable attention from the research community. However, despite its key role, there is a common consensus that OCL is the least adopted among all languages in the UML. It is often argued that, software practitioners shy away from OCL due to its unfamiliar syntax. To ensure better adoption of OCL, the usability issues related to producing OCL statement must be addressed. To address this problem, this paper aims to preset a method involving using Natural Language expressions and Model Transformation technology. The aim of the method is to produce a framework so that the user of UML tool can write constraints and pre/post conditions in English and the framework converts such natural language expressions to the equivalent OCL statements. As a result, the approach aims at simplifying the process of generation of OCL statements, allowing the user to benefit form the advantages provided by UML tools that support OCL. The suggested approach relies on Semantic Business Vocabulary and Rules (SBVR) to support formulation of natural language expressions and their transformations to OCL. The paper also presents outline of a prototype tool that implements the method.**

Keywords- Natural languages, SBVR, OCL, Model Driven Development,

I. INTRODUCTION

Unified Modeling Language (UML) is now considered as the de-facto standard for software modeling. One of the languages in UML is Object Constraint Language (OCL). OCL is widely used in expressing constraints and well-definedness in the UML standards. OCL can be used as part of UML model to significantly improve the clarity of software models and make models more precise [26]. In addition, OCL is well supported with numerous tools [32], [33], [19], [34], [35] that provide type checking support. However, there is a common consensus that the least adopted member of UML family of languages is OCL. Indeed, software practitioners shy away from the OCL mostly due to unfamiliar syntax and semantics. In complex models, writing correct OCL statements is non-trivial; it is often argued that manual effort to create an OCL constraint usually results in inaccurate and erroneous constraints specification [25], [27]. In order to benefit from the OCL, the usability aspects of the language must be addressed.

The idea of this paper is motivated by recent progresses in *Model Driven Development* (MDA) [3] and birth of *Semantic Business Vocabulary and Rules* (SBVR) [13]. The presented approach allows the user to write various constraints and pre/post conditions on a UML model in natural languages [14] (NL) e.g. English. The NL specification of the constraints is automatically transformed to an equivalent OCL syntax. Users can be assisted to write better OCL statements in shorter time by using a NL based user interface. A tool support can be used for automatic translation of the NL to OCL statements. But, formalizing of the concepts in natural language representation to a clear and unambiguous mode is a real challenge. SBVR can be useful in achieving a clear and unambiguous representation. In NL to OCL transformation, the use of SBVR not only makes the NL easy to semantically analyze but also it provides OCL resembling syntax.

The objective of the paper is to improve the OCL usability by writing constraints in a natural language and then generating the OCL constraints from NL specification by doing automated transformation. Automated transformation is used to hide the complexity involved in the manual production of OCL constraints from NLs. Model transformations will provide a systematic and attributed way of creating OCL from NL but also results in producing OCL statements in a seamless and non-intrusive manner.

The paper is organized as follows. Next section describes preliminary concepts related to the Model Transformation, Natural Languages, SBVR, and OCL. Section 3 describes the problem addressed in the paper. Section 4 presents a sketch of the solution and describes model transformation from Natural Languages to SBVR and then from SBVR to OCL. Section 5 illustrates the implementation, followed by a discussion on the related work. The paper ends with a conclusion section.

II. PRELIMINARIES

A. Object Constraint Language (OCL)

OCL [18] is a formal language used to annotate a UML model with the constraints. The typical use of OCL is to represent functional requirements using class invariants, pre and post conditions on operations and other related expressions on a UML model [19]. OCL can also be used for representing non-functional requirements [20].

1541-7719/10 $26.00 © 2010 IEEE
DOI 10.1109/EDOC.2010.33

OCL abstract syntax defines the grammar and structure of an OCL statement. The OCL abstract Syntax is further defined into OCL types and OCL expressions. Common OCL types are data types, collection types, message types, etc. While, OCL expressions can be *property* expression, *if* expression, *iterator* expression, *variable* expression, etc. We have used a selected set of OCL abstract syntax for implementation. Figure 1 shows the abstract syntax metamodel [18] of selected OCL expressions, we refer the reader to OCL 2.0 document [18] for further information on OCL.

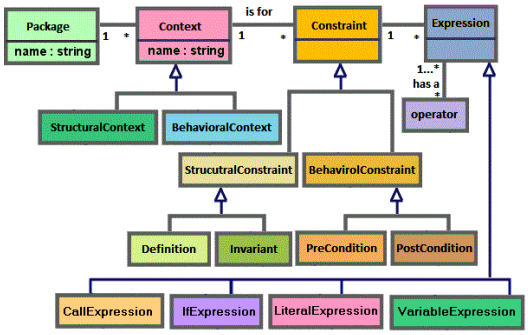
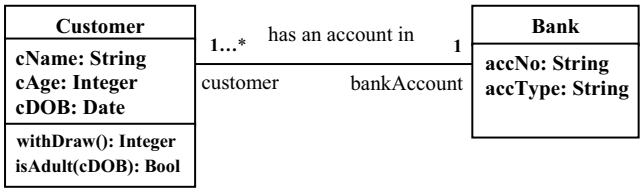


Figure 1. Elements of selected OCL meta-model

While creating UML models such as Class diagrams for expressing structure of a system or sequence diagram for describing interaction between objects is almost intuitive, writing OCL expressions requires not only an adept knowledge of OCL syntax but also skilful understanding of the semantics of OCL expressions. Consider a scenario involving a bank, where a customer has a bank account as depicted in figure 2.




```
context    customer
inv :      self.cAge >= 18
```

Figure 2. Example conceptual schema of a bank account

In this paper, we target generation of OCL constraints. A constraint is a restriction on state or behaviour of an entity in a UML model [19]. The OCL constraint defines a Boolean expression. If the constraint results true, the system is in valid state. Consider the following three categories of the OCL constraints:

1) Invariants: The invariants [3] are conditions that have to be TRUE for each instance of the model.

```
context customer
inv: self.cAge >= 18
```

2) Precondition: A precondition [3] is a constraint that should be TRUE always before the execution of a method starts.

```
context customer :: isAdult(cDOB:
Integer): Boolean
pre: self.cDOB >= 1990
```

3) Postcondition: A postcondition [3] is a constraint that should be TRUE always after the execution of a method has finished.

```
context driver :: isAdult(cDOB: Integer):
Boolean
post: result >= 18
```

In this paper, we propose a natural language (e.g. English) based user interface for writing constraints for a UML model. Natural languages are ambiguous and unclear; we propose the use of Semantic Business Vocabulary and Rules (SBVR) standard to deal with the syntactical inconsistencies and semantical ambiguities involved in the NL representation. The transformation from NL to OCL involves two stages. Firstly, NL specification is automatically transformed to SBVR representation and then finally to the OCL constraints.

B. *Semantic Business Vocabulary and Rules*

Semantic Business Vocabulary and Rules (SBVR) [13] is a recently introduced standard by OMG. Using SBVR, specifications can be captured in natural languages and represented in the formal logic so that they can be machine-processed. Figure 3 shows SBVR metamodel:

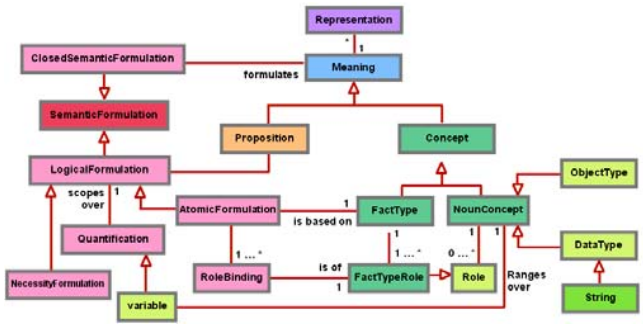


Figure 3. – Elements of selected SBVR meta-model

SBVR representation has two major elements: SBVR vocabulary and SBVR Rules. Brief details of these two major elements of SBVR are given below.

1) SBVR Vocabulary: SBVR vocabulary is based on two elements: Concepts and Fact Types. A concept is a key term that represent a business entity in a particular domain. There are two common types of concepts [13]: noun concept and individual concept. Typically, the common nouns are classified as noun concepts while the proper nouns or quantified nouns are denoted as individual concepts. A fact type is a verb, a proposition, or a combination of both [13].

A Fact type specifies the relationship among different concepts in a business rules.

An example of SBVR rule for UML model shown in figure 1 can be as "**It is necessary that each** customer *should be* **at least 18** years old". In this example, "**It is necessary that**" is a SBVR keyword, "**each**" is a keyword and a quantifier, "customer" is a noun concept, "*should be*" is verb and "**at least**" is SBVR keyword, 18 is a quantifier, and "years" is a noun concept. In this example, "customer *should be* old" is a fact type.

2) SBVR Rules: SBVR propose use of the SBVR rules to represent particualr business logic in a specific context. The SBVR rules can be of two types [13]: definitional rules and behavioral rules:

- Definitional Rules or structural rules are used to define an organization's setup [13] e.g. **It is necessary that each** customer *has* **at least** one *bank account*.
- Behavioural Rules or operative rules express the conduct of an entity [13] e.g. **It is obligatory that each** customer *can withdraw* **at most** GBP 200 per day.

3) Formalizing NL Text Representation. In SBVR 1.0 document [13], the Structured English is proposed, in Annex C, as a possible notation for the SBVR rules. The Structured English provides a standardized representation to formalize the syntax of natural language representation. In this paper, we have used the following Structured English specification:

- noun concepts are underlined e.g. customer
- *verbs* are italicized e.g. *should be*
- **keywords** are bolded i.e. SBVR keywords e.g. **each**, **at least**, **at most**, **obligatory**, etc.
- individual concepts are double underlined e.g. silver account customer

Here, we purpose a new element adjective. We have dotted underlined the adjectives e.g. in the above given example, "old" is an adjective. The adjectives are used to identify the attributes.

4) Formulating NL Text Semantic: Logical formulations are used to semantically formulate the SBVR rules. Common logical formulations are [13]:

- Atomic formulation specifies a fact type in a rule e.g. "customer *should be* old" is atomic formulation from the fact type "customer *is* old".
- Instantiation formulation denotes an instance of a class e.g. "silver account" is an *Instantiation* of the noun concept "bank account".
- Logical operations e.g. conjunction, disjunction, implication, negation, etc are also supported in SBVR. In natural languages, the logical operations allow combining NL phrases to create more complex logical expression.
- Quantification states the enumeration of a noun concept or verb concept e.g. "at least one", "at most one", "exactly one", etc are used to quantify concepts.
- Modal Formulation identifies the meanings of a logical formulation. e.g. "It is obligatory" or "It is necessary" are used to formulate modality.

For mapping the natural languages with SBVR, the NL phrases are mapped with SBVR vocabulary and particular meanings of a NL phrase are mapped using SBVR rules.

C. Model Transformation

Model Driven Architecture (MDA) [1] is a flavor of model-driven development (MDD) [20] proposed by the OMG . Central to the MDD and MDA is the process of model transformation, i.e. automated creation of new models, which is depicted in Figure 4 and can be described briefly as follows. Model Transformations rely on the "instanceof" relationship between models and metamodels to convert models. Model Transformations define the mappings rules between two modelling languages metamodels. Rules typically define the conversion of element(s) of the source metamodel to equivalent element(s) of the destination metamodel. The Model Transformation frameworks execute the Model Transformation implementations on models. Upon execution with a given model, the necessary rules are applied by the transformation framework, applying rules to generate an equivalent model in the destination modelling language.

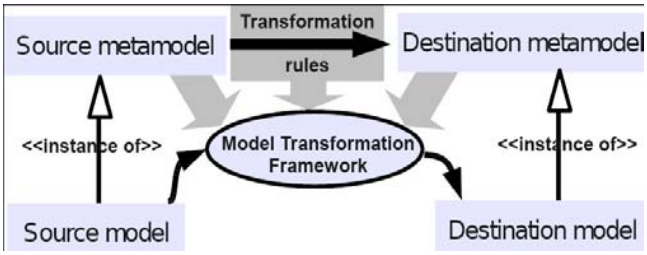


Figure 4. : An overview of MDD

There are different types of model transformations such as model-to-model, model-to-text and text-to-model transformations [7]. The *Model-to-Model Transformation* is used to transform a model into another model e.g. translating OCL to SBVR [8], UML to SBVR [8], and SBVR to UML [9]. The *Model-to-Text Transformation* is used to translate a model to a natural language representation e.g. transforming OCL to NL [10], and UML to NL [11]. The *Text-to-Model Transformation* talks about interpreting the natural language text and create a model from the interpretation.

In our approach, we propose the use of model-to-model transformation for automated transformation of NL to OCL constraints. A Typical model transformation is employed by creating abstract syntax of source model and then converting it into the target model representation using the model transformation rules. We have used a set of transformation rules to perform the proposed transformation of NL to OCL.

III. DESCRIPTION OF THE PROBLEM

UML has been adopted as the de facto standard for the design, modeling and documentation of software systems [19]. There are lots of tools which not only allow modeling and design but also allow support for creation of code, reverse engineering, versioning and many more [24]. However, it is a well know fact that the least used of all UML languages is OCL. This is often attributed to complex syntax of OCL [7].

The ability of using OCL by the developer is very important. Correctly written OCL constraints and pre/post conditions improve the clarity of software models significantly and make models more precise [26]. Manual effort to create an OCL constraint may result in inaccurate and inconsistent constraints specification [25], [27]. We can increase OCL usability through automatically generating accurate and consistent code for OCL constraints. Besides, constraints specification, OCL can be used for specifying models for analysis purposes, such as, among others, our work on transformation of UML to Alloy [28]. Improving the usability of OCL will also assist developers who are not experts of formal methods for producing specifications in other languages e.g. Z, B, Petri-nets, for automatic analysis.

This paper plans to present a technique that allows development of tools and techniques assisting in writing OCL. Wahler's research has addressed this problem by using template based approach [27]. Kristofer J. presents a rule system [21] to transform an OCL expression into NL representation. Similarly, Linehan presented his work for the translating Structured English representation to predicate logic [25] and then finally this mathematical representation is transformed into equivalent Java structures. However, we are adopting a radically new approach by bringing together two main domain of computer science: model transformation and Natural language through adopting SBVR. Using natural languages and transformation to OCL seems like an intuitive approach. However, we adopt a systematic way to use SBVR to restrict the domain of NL text and generate OCL code from the SBVR representation.

The objective of this paper is to present a method of using SBVR for better formulation of natural languages and then allow development of tools for better writing of OCL statements from SBVR based natural language text.

IV. SKETCH OF THE SOLUTION

In this section, we present a framework to perform NL to SBVR and SBVR to OCL transformation. In the first transformation, the preprocessed English language text is transformed into a conceptual model in SBVR. The second transformation is performed to translate the SBVR expressions into OCL statements. Figure 5 highlights the sketch of stages involved.

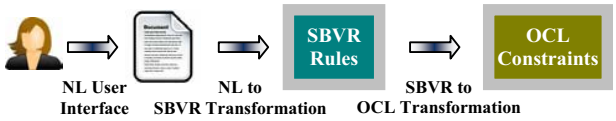


Figure 5. Application Scenario of NLto OCL transformation

A. NL to SBVR Transformation

The transformation of natural language specification to SBVR rules is performed by performing following four steps:

1) Parsing NL Specification
2) Extracting Fact Types
3) Verifying with the UML Model
4) Apply Semantic Formulation

A breif description of all these steps is provided in the following section.

1) Parsing NL Specification: The NL represeantaiton is semanically analyzed using LESSA [14] (Language Engineering System for Semantic Analysis) approach. Parsing information of a SBVR rules "A customer should be 18 years old" is following:

TABLE I. NATURAL LANGUAGE ABSTRACT SYNTAX INFORMATION.

Syntactic Elements	Meanings	Syntactic Elements	Meanings
S	English Sentence	V	Action Verb
NP	Noun Phrase	DT	Determiner
VP	Verb Phrase	AJ	Adjective
VB	Verb	N	Noun
P	Pronoun	NR	Number
HV	Helping verb	PP	Preposition Phrase

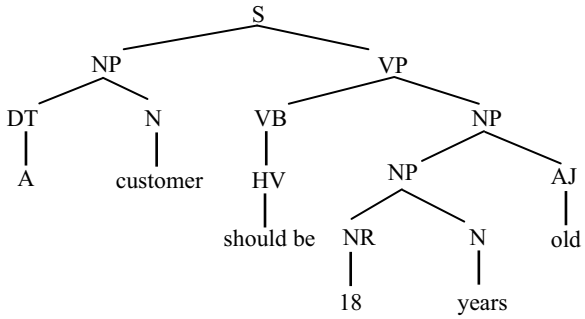


Figure 6. Parse information of natural language text

2) Extracting Fact Types: The generated NL semantic information is further mapped with SBVR syntax via a set of transformation rules, which are described in section 5.1. To explain briefly, in this phase the noun concepts and verbs are identified from the semantic information. A set of transformation rules have been used to identify the noun concepts, verbs, and attributes from the NL text and map with the target SBVR rules. Finally the fact types are formed from the extracted noun concepts, verbs, and attributes. Transforamtion rules to extract fact types are based on the following facts:

- Common names are transformed noun concpets
- Proper nouns are transformed into individual concpets
- Auxilary verbs and action verbs are transformed into *verbs*
- Adjectives are used to produce the *attributes*.

To find the individual concepts, the instantiation formulation is employed. An instantiation formulation is equivalent to existential quantification and uses only one concept at a time [13]. Here, the concept is bound with a bind-able target that is similar to the process of instantiation in OO and UML e.g. "John is a customer." can be formulated as below:

. The instantiation formulation considers the concept "customer."
. The instantiation formulation binds to the individual concept 'John'.

3) Verifying with the UML Model: Before generating a SBVR rule from NL text, the extracted information from NL text is semantically verified with the given UML model. The semantic verification is performoed by mapping the noun concepts, individual concepts, adjectives, and verbs from NL text with the class names, instance names, attribute names and method names in the given UML class model. This semantic verification is performed to verify that the input NL text is semanticly related to the target UML model for which the OCL constraint will be generated. Following is the example of the UML model verified SBVR rule:

It is necessary that each customer *must be* 18 years cAge.

4) Applying Semantic Formulation. SBVR standard 1.0 [13] has defined a set of logical formulations to devise the natural language text in a structured and consistent manner. For the different types of syntactic structures used in English language, respective types of logical formulations have been defined. Following are the details that how we have incorporated these logical formulations to map English language text into SBVR metamodel.

a) Atomic Formulation: An atomic formulation is based on only one fact type and a fact type role is bind with the respective fact type [13]. A NL statement e.g. "The customer has silver account" can be atomic formulated by with a fact type 'customer has account'. The process of atomic formation defiend in SBVR 1.0 is as following:

. The atomic formulation is based on the fact type 'customer has account'.
. The atomic formulation has a first role binding.
. . The first role binding is of the role 'customer' of the fact type.
. . The first role binding binds to the individual concept 'The customer'.
. The atomic formulation has a second role binding.
. . The second role binding is of the role 'account' of the fact type.
. . The second role binding binds to the individual concept 'silver account'

For the further reading we recommend to the reader SBVR 1.0 document, section 9.2.2 [13].

b) Logical Operations. Logical operations are used to combine one or more expressions, known as logical operand to produce complex Boolean expressions [13]. We have incorporated these logical operations to map NL phrases to more complex logical expression. We are currently supporting the following six types of the logical expressions which are defined in SBVR v1.0 [13] document.

i. *Conjunction* is a binary operation for logical decision of two operands to formulate the meanings that each operand is true i.e. *p* AND *q*
ii. *Disjunction* is a binary operation for logical decision of two operands to formulate the meanings that at least one operand is true i.e. *p* OR *q*
iii. *Equivalence* is a binary operation for logical decision of two operands to formulate the meanings that both operands are true of false i.e. *p* is equal to *q*
iv. *Implication* is a binary operation for logical decision to formulate the meanings that second operand is true if first operand is true i.e. if *p* then *q*.
v. *Negation* is a unary operation for logical decision of one operand that formulates the meanings that the operand is false i.e. NOT *p*.

c) Quantifications. Quantification is a logical formulation that uses a variable to specify the scope of a concept. Four basic types of quantifications have been defined in SBVR v1.0 [13]. Quantification types are briefly described below:

i. *At least n quantification*: An existential quantification shows min. cardinality
ii. *At most n quantification*: This quantification shows max. cardinality
iii. *Numeric range quantification*: It exhibits both min. and max. cardinality
iv. *Exactly n quantification*: This quantification shows the exact cardinality.

d) Modal Formulations. Modal formulations are logical formulations that are used to specify meanings of the other logical formulations. There are four basic types of modal formulations [13].

i. *Necessity Formulation*: if a logical formulation is true in all possible worlds.
ii. *Obligation Formulation*: if a logical formulation is true in all acceptable worlds.
iii. *Permissibility Formulation*: if a logical formulation is true in acceptable worlds.
iv. *Possibility Formulation*: if a logical formulation is true in some possible worlds.

Following example highlights basic types of logical formulations in a SBVR rule.

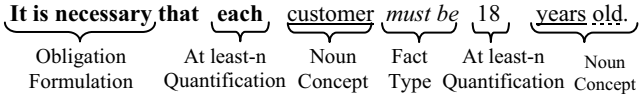


Figure 7. Logically formulating a SBVR rule

B. SBVR to OCL Transformation

OCL code is created by generating different fragments of the OCL expressions and then concatenating the created fragments to compile a complete OCL statement. The following section describes the process of creation of abstract syntax model for OCL constraints. Following steps

were performed for NL to SBVR transformation.

1) Extracting Classes, Methods and Attributes from SBVR rules.
2) Generation of OCL expressions from the extracted information e.g. Classes, Methods and Attributes
3) Mapping OCL Syntax
4) Mapping OCL Semantics

1) Extracting Classes, Methods and Attributes: The first step is to identify the noun concepts, individual concepts, verbs, attributes and fact types in the source SBVR rule. In a SBVR rule the nouns concepts represent a class, the individual concepts represent an instance of a class, the verbs represent methods of a respective class or instance, and the fact type represents a relationship a UML model. The adjectives represent the attribute of a class or an instance. Following information is employed to extract classes and their respective methods and attributes from the SBVR rules:

- Each noun concepts are mapped into classes.
- Each individual concepts are mapped as instances of a class which are identified from the noun concepts
- Each *verb* is mapped into a method of the respective class.
- The adjectives are mapped as attributes of a class or an instance.

2) Generating OCL Expression: The extracted classes, methods and attributes are used to find OCL context, invariant body, logical expressions and collection expressions. These OCL elements were combined to make a complete OCL expression. Following steps are performed to generate an OCL expression.

Source: **It is necessary** that **each** customer *should be* **at least 18** years old.

Step I – The noun concept is mapped with the context i.e. customer or bank e.g. `context customer`

Step II – Adding the instance i.e. bank customer e.g. `customer or self`

3) Mapping OCL Syntax. In the context of our research domain, OCL syntax rules will help to extract the desired information. To translate a SBVR statement into OCL expression, an OCL abstract syntax model is designed that is based on OCL version 2.0 [18]. The OCL abstract syntax model is shown in Table 4.3**.**

TABLE II. OCL SYNTAX MODEL

Syntax Rule		Syntactic Elements
Constraint	→	**Context** *Context-Name* **inv**: *Constraint-Body*
Context-Name	→	*Identifier* | *Identifier* :: *Identifier* [: *Class*]
Identifier	→	Context | Context : Method-Name (*Parameters*): *Literal*
Class	→	*Identifier* :: *Identifier* | *Collection-Name* (*Class*)
Constraint-Body	→	*Expression* | (**pre** | **post**) : *Expression*
Expression	→	**If** *Expression* **then** *Expression* **else** *Expression* **endif** | Expression (. | →) *Method* | *Expression InfixOper Expression* | *PrefixOper Expression* | *Literal*
Method	→	**forAll** | **exists** | **select** | **allInstances** | **include** | **iterate** | …
Parameters	→	Parameter-Name : *Literal*
Literal	→	**Integer** | **Real** | **String** | **Boolean** | **Collection**
Collection-Name	→	**Collection** | **Set** | **Bag** | **Sequence**
InfixOper	→	**+** | **-** | ***** | **/** | **=** | **>** | **<** | **>=** | **<=** | **<>** | **OR** | **AND** | **XOR**
PrefixOper	→	**-** | **not**

In the OCL syntax mapping phase, the extracted OCL constitutes are concatenated according to the OCL abstract syntax given in table II. Here, we use the constraint rule and replace the extracted context with the *Context-Name* and generate the following expression:

```
context   customer
inv:   Constraint-Body
```

In the above example, the *Constraint-Body* is generated in the OCL semantic mapping phase.

4) Mapping OCL Semantics. Logical formulations were defined that were based on the concrete structures of the OCL constraint expressions: operators, binary operations, implication rules, constraints, and collections as given below:

TABLE III. OCL OPERATORS FOR SBVR CONSTRUCTS

Logical Formulation	OCL	Logical Formulation	OCL
Structure SBVR rule	`a.b`	b is *data type* of a	`a:b`
Behavioral SBVR rule	`a•b`	b is *return type* of a	`a():b`
method a of class b	`a::b`	comments	`- -`

TABLE IV. BOOLEAN AND ARITHMETIC OPERATIONS OF OCL

Logical Formulation	OCL	Logical Formulation	OCL
b1 is *less than* b2	`b1<b2`	b1 is *added* with b2	`b1+b2`
b1 is *less than* OR *equals to* b2	`b1<=b2`	b1 is *subtracted* from b2	`b1-b2`
b1 is *greater than* b2	`b1>b2`	b1 is *multiplied* with b2	`b1*b2`
b1 is *greater than* OR *equal to* b2	`b1>=b2`	b1 is *divided* by b2	`b1/b2`
b1 is *equal to* b2	`b1=b2`	b1 is *not equal* to b2	`b1<>b2`

TABLE V. OCL LOGICAL STATEMENTS

Logical Formulation	OCL	Logical Formulation	OCL
a is T and b is T	`a AND b`	a is T or b is T: any one	`a or b`
b is T then T	`a implies b`	a is T or b is T: not both	`a xor b`
a is F	`Not a`		

TABLE VI. OCL INVARIANTS AND OTHER RELATED EXPRESSIONS

Logical Formulation	OCL	Logical Formulation	OCL
a is *context*	`Context a`	Variable's initial value	`init`
Constraint/*Invariant*	`Inv`	Derive variable	`Derive`
If TRUE *then* a *else* b	`If T then a else b`	Define public variable	`Define`
Group classes context	`Package`	Define private variable	`Let`
Pre-condition	`Pre`	operation post-condition	`Post`

TABLE VII. OCL COLLECTIONS

Logical Formulation	OCL
More than One Elements	`Collection(T)`
Unique AND *Unordered* list of elements	`Set(T)`
Unique AND *Ordered* list of elements	`OrderedSet(T)`
Non-Unique (Repeating) AND *Unordered* list of elements	`Sequence(T)`
Non-Unique (Repeating) AND *Ordered* list of elements	`Bag(T)`

In the OCL semantic mapping phase, the defined logical formulations in table III to table VII were mapped with the target OCL statements. Here, we generate the constraint body and replace it with the *Constraint-Body* parameter in the constraint rule used in the syntax mapping phase. As the source SBVR rule is structure rule, hence we use `a.b` OCL construct and replace *a* with the class name or `slef` Keyword and *b* is replaced with the attribute name to generate the following expression:

```
inv:  self.cAge
```

Here, the OCL expression `self.cAge` is yet to be evaluated with the logical condition. The logical condition is created by transforming the quantification 'less than or equal to 18' to the OCL Boolean operation `self.cAge >=18` and complete the OCL expression as following:

```
context   customer
inv:  self.cAge >= 18
```

V. A SKETCH OF IMPLEMENTATION

In this research, the proposed framework for NL to OCL transformation is based on model transformation. This section presents how model transformation rules have been employed to translate natural language text to OCL code.

A typical model transformation is carried out by using a rule based approach to translate source text or a model conforming to its metamodel into a target text or model conforming to its metamodel [3]. Rule based model transformations employee set of transformation rules to map source model to a target model. A transformation rule *r* maps one component of the source model using a source transformation rule r_s with the one component of target model using a target transformation rules r_t. We can represent it as r: S → T [12].

Transformation rules were individually defined for both parts of NL to OCL transformation: NL to SBVR and SBVR to OCL. Defined transformation rules were based on If-then-Else structure [4]. Each rule consists of a component from the source model (NL or SBVR) and one component from the target model (SBVR or OCL) inspects source input and if the mapping [8]. We have defines a number of states for the source model, e.g. $Y = \{y_1, y_2, \ldots, y_n\}$ is a set of states for source model. Similarly, a number of states for the target model have been defined, e.g. $Z = \{z_1, z_2, \ldots, z_n\}$ is a set of states for target model. For mapping, the states of input source model are matched with possible states of the target model. An occurrence of X from the source model is looked within the all occurrences of Z from the source model and if the match is found, the matched state of source model is given concrete syntax of the target model. The following section elaborate how a SBVR rule is transformed to OCL constraint using transformation rules as follows:

Step I – **It is necessary** that **each** customer *should be* **at least 18** years old.

Step II – **It is necessary that <**Quantification**>** <actor> *should be* <Quantification> years <attribute>.

This generalized representation is finally transformed to the OCL constraint by using the defined transformation rules. A typical model transformation rule comprises of the variables, predicates, queries, etc [17]. A transformation rule consists of two parts: a left-hand side (LHS) and a right-hand side (RHS) [11]. The LHS is used to access the source model element, whereas the RHS expands it to an element in the target model. The transformation rules for each part of the OCL constraints are based on the abstract syntax of SBVR and OCL that are given in the following section. Rule 1 transforms the noun concept (actor) in SBVR rule to OCL context for an invariant and the Rule 2 transforms the noun concept (actor) and *verb* (action) in SBVR rule to OCL context for a pre/post condition.

Rule 1

T[*context-inv*(actor)] = *context-name*

Rule 2

T[*context-cond*(actor, action)] = *context-name* **::** *operation-name*

To generate the body of an invariant and pre/post-conditions a complete set of rules were defined. Due to shortage of space we present few of them. Here the rules 3 transforms SBVR information to the OCL invariant, while rule 4 and 5 are used to transform SBVR rules to OCL preconditions and post conditions.

Rule 3

T[*invariant*(*context-inv, inv-body*)]
= `context` *context-inv*\
`inv:` *inv-body*

Rule 4

T[*pre-cond* (*context-cond, pre-cond-body*)]
= `context` *context-cond*
`pre:` *pre-cond-body*

Rule 5

T[*post-cond* (*context-cond, post-cond-body*)]
= `context` *context-cond*
`post:` *post-cond-body*

These are some of the rules that were used for SBVR to OCL transformation. Our approach only addresses the invariants and pre/post conditions. We have not addressed yet the OCL queries.

VI. TOOL SUPPORT

OCL-Builder tool was implemented to translate SBVR to OCL constraints. Translation rules and the abstract syntax of OCL and SBVR were implemented in .NET platform using Visual Basic Language. LESSA [22] syntax analyzer was used to syntactically analyze SBVR rules and map them with given UML class Model in XMI 2.1 format. OCL-Builder tool then finally translates the UML-SBVR specification to OCL constraints by using translation rules. Figure 8 shows a screen-shot of OCL-Builder tool.

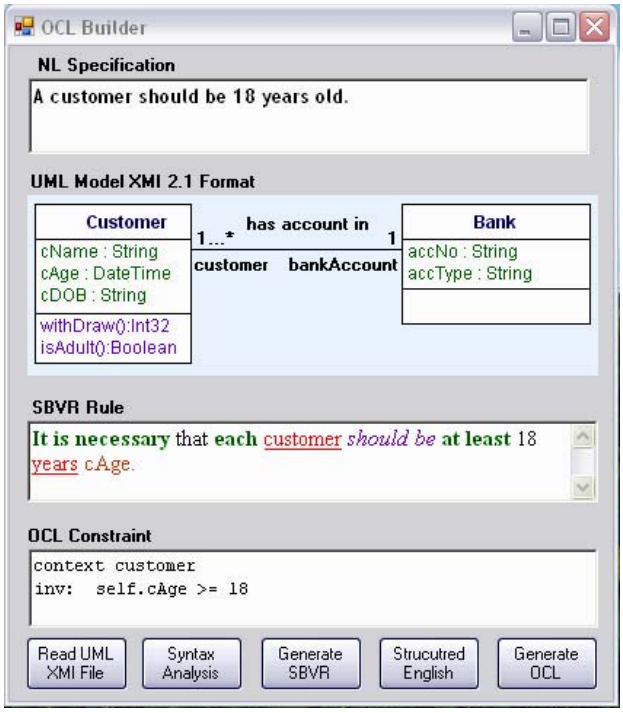


Figure 8. OCL-Builder tool – Transforming SBVR to OCL

OCL-Builder tool generates one OCL constraint for one UML-SBVR specification. SBVR rule is transformed to OCL constrain in five steps. First of all, a XMI file is read as input for the target UML class model. Then, the input SBVR rule is syntactically analyzed and mapped with the given UML class model. All the classes and associations in given UML class model are mapped to the input SBVR rule. The output of this activity is UML mapped SBVR rule that is named UML-SBVR specification. Then, the Structured English formatting is applied in it. The last step is the generate OCL from the UML-SBVR specification.

VII. RELATED WORK

In the last decade, a number of software tools has been designed and implemented to facilitate OCL code parsing and validation. Common examples of such OCL tools are Dresden OCL Toolkit [32], IBM OCL Parser [33], USE [19], ArgoUML [34], Cybernetic OCL Compiler [35], etc. But these tools are limited to verify the syntax and type checking of the already written OCL code. Currently, no suitable tool is available that is capable of automatic generation of OCL code from natural languages. This is the major reason that after more than ten years of introduction of OCL, OCL is yet no fully adapted in software developer communities as other language of UML. The related work is briefly expressed in reaming part of the section.

In last decade, many automated solutions for transformation of natural language software requirement specification (SRS) to UML based formal presentation have been presented [36], [37], [38], [39], [40]. Introduction of frameworks and tools for automated transformation NL to UML model have made things very easy and time saving for the software designers.

J. Cabot also presented some transformation techniques to get semantically alike representations of an OCL constraint [7]. The proposed technique assists in simplifying the modeling phase of software development by increasing the understanding level of the designer by providing him more than one alternate OCL representations. The presented work can also help out in future for PIM-to-PIM, PIM-to-PSM and PIM-to-code transformations. In the same direction of research, Bryant presented a system that used natural language processing for developing a business domain model from system requirements given in natural languages [41]. First of all simple pre-processing actions were performed on the input text i.e. spelling and grammar checking, use of consistent vocabulary and appropriate sentence usage. Afterwards this pre-processed text was translated into XML representation with the help of domain specific knowledge. Bryant has also proposed that business domain model created by his work can be further transformed into UML representation.

The transformation of formal specifications (e.g. OCL) to information specifications (e.g. NL) is another recent trend of research regarding OCL. Transformation of OCL into NL has been addressed by Kristofer J. where he presented a rule system. The rule system annotates an OCL syntax tree with OCL semantic annotation on it and also performs disambiguation of syntax trees [21]. This work is part of the project for translating OCL code into a natural language representation e.g. English or German. The major limitations of this work are that it supports OCL 1.5. J. Linehan presented his work for the translating Structured English representation to predicate logic [25] and then finally this mathematical representation is transformed into equivalent OCL structures. In the context of our proposed research, his work is not helpful because, writing SBVR rules in Structured English representation is itself an overhead.

On the other hand, some work has been done to transform OCL and UML to SBVR by Cabot [8]. He proposed automatic transformation of UML and OCL schema to SBVR specification. This work is basically reverse engineering of software modeling and better for generating

business vocabularies from the already designed software models. In the akin trend, Amit presented his work to transform SBVR business design to UML models [9]. He has used model driven engineering approach to transform SBVR specification into different UML diagrams e.g. activity diagram, sequence diagram, class diagram. His research work is a millstone in reducing gap between SBVR and UML based software modeling.

All the related work, highlights that lot of work has been done in the area of proposed research but the presented area is yet unaddressed. The related work also clarifies the pressing need of a mechanism that is capable of providing assistance for OCL coding.

VIII. Conclusions and Future Directions

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 Natural Language expressions. The presented transformation makes use of SBVR as an intermediate step to highlight the syntactic elements of natural languages 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. The presented method is implemented as prototype tool which is being extended to be integrated into the existing tools. As a next step, we are hoping to investigate usability aspects of the tool directly via empirical methods involving teams of developers.

References

[1] Warmer Jos, Kleppe A.: The Object Constraint Language – Getting Your Models Ready for MDA. Second Edition, Addison Wesley (2003)

[2] Ceponiene L., eta al: Separation of Event and Constraint Rules in UML & OCL Models of Service Oriented IS. Information Tech. and Control, 38 (1), pp. 29--38 (2009)

[3] Dang D., Gogolla M., Precise Model-Driven Transformations Based on Graphs and Metamodels, 7th IEEE International Conference on Software Engineering and Formal Methods 23-27 November, 2009, Hanoi, Vietnam, p:307-316 (2009)

[4] Gogolla M., Büttner F., Dang D., From Graph Transformation to OCL Using USE. The third International Workshop and Symposium on Applications of Graph Transformation with Industrial Relevance, pp.585--586 (2007)

[5] Linehan M.: SBVR Use Cases. In International Symposium on Rule Representation, Interchange and Reasoning on the web, RuleML, LNCS Vol. 5321 pp. 182--196 (2008)

[6] Tae Y. K., Yun K., Heung S.: Towards Improving OCL-Based Descriptions of Software Metrics. In 33rd IEEE COMPSAC, pp. 171--179 (2009)

[7] Cabot J., Teniente E.: Transformation Techniques for OCL constraints, J. of Science of Computer Programming, 68(03) Oct 2007, p.152--168 (2007)

[8] Cabot J., et al.: UML/OCL to SBVR Specification: A challenging Transformation, Journal of Information systems doi:10.1016/j.is.2008.12.002 (2009)

[9] Raj A., Prabharkar T., Hendryx S.: Transformation of SBVR Business Design to UML Models. In ACM Conference on India software engineering, pp.29--38 (2008)

[10] Burke D., Kristofer J.: Translating Formal Software Specifications to Natural Language. Springer LNCS, Vol. 3492, pp. 51--66 (2005)

[11] Raquel R., Cabot j.: Paraphrasing OCL Expressions with SBVR, 13th International C. Natural Language and Information Systems: Applications of NL to IS, pp.311--316, (2008)

[12] Bajwa I. S., Hyder I., "UCD-Generator - A LESSA Application for Use Case Design", Proceedings of IEEE- International Conference on Information and Emerging Technologies- ICIET, pp-182-187 (2007)

[13] OMG: Semantics of Business vocabulary and Rules (SBVR), OMG Standard, v. 1.0, (2008)

[14] Spreeuwenburg S., Healy K.: SBVR's Approach to Controlled Natural Language. Workshop on Controlled Language – CNL 2009, Marettimo, Italy (2009)

[15] Kleiner M.: ATL Parsing SBVR-Based Controlled Languages, LNCS – Model Driven Engineering and language systems, Vol. 5795, pp.122--136 (2009)

[16] Mathias K., Patrick A., Bezivin J.: Parsing SBVR-Based controlled Languages, In Models'09 Colorado USA, p: 122-136, (2009)

[17] Silvie S., Keri A., SBVR's Approach to Controlled Natural Language, Workshop on Controlled Natural Language 8-10 June, 2009, Marettimo Island, Italy (2009)

[18] OMG: Object Constraint Language (OCL), OMG Standard, v. 2.0, (2006)

[19] Gogolla M., et al.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming, Vol. 69 pp. 27--34 (2007)

[20] Brucker A., Doser J., Wolff B.: An MDA Framework supporting OCL. In Electronic Communications of EASST, 2006, Vol. 5, pp. 2--19 (2006)

[21] Kristofer J.: Disambiguation Implicit Constructions in OCL. In Conference on OCL and Model Driven Engineering, October 12, 2004, Lisbon, Portugal, pp. 30--44 (2004)

[22] Yu, F., Bultan, T., and Peterson, E.: Automated size analysis for OCL. In ACM Symposium on the Foundations of Software Engineering, pp. 331--340, (2007)

[23] Warmer Jos, Kleppe A.: The Object Constraint Language – Getting Your Models Ready for MDA. Second Edition, Addison Wesley (2003)

[24] Engels G., Heckel R., K¨uster J.: Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model, LNCS Vol. 2185, pages 272--287 (2001)

[25] Linehan M.: Ontologies and rules in Business Models. In 11th IEEE EDOC Conference Workshop, pp. 149-156, (2008)

[26] Meyer. B.: Object-Oriented Software Construction. International Series in Computer Science, Second Edition, Prentice-Hall (1997)

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

[28] Shah A., Anastasakis K., Bordbar B.: From UML to Alloy and Back, In ACM International Conference Proceeding Series, Vol. 413, pages 1--10 (2009)

[29] Scott W. Ambler. The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge University Press, 3rd Edition, 2004.

[30] Linehan M.: Semantics in Model Driven Business Design. In 2nd International Conference on Semantic Web Policy Workshop, Athens, GA, USA, pp.1--8 (2006)

[31] Cate T., Kolaitis P.: Structural characterizations of schema-mapping languages. In 12th International Conference on Database theory, Vol. 361, pp.63--72 (2009)

[32] Demuth B, Wilke C.: Model and Object Verification by Using Dresden OCL. In R.G. Workshop on Innovation Information Technologies: Theory and Practice, pp. 81--89 (2009)

[33] IBM OCL Parser, Sep 2009 http://www-01.ibm.com/software/awdtools/library/standards/ ocl-download.html, (2009)

[34] Bart V. R., et al.: On the Detection of Test Smells: A Metrics-Based Approach for General Fixture and Eager Test. IEEE T. on Software Engineering 33(12) pp. 800--817 (2007)

[35] Emine G. A., Richard F. P., Jim W.: Evaluation of OCL for Large-Scale Modelling: A Different View of the Mondex Purse. Springer LNCS Vol. 5002, pp. 194-205 (2008)

[36] Ilieva M., Olga O.: Automatic Transition of Natural Language Software requirements Specification into Formal Presentation. Springer LNCS Vol. 3513, pp.392--397 (2005)

[37] Whittle J., Jayaraman P., et al.: MATA: A Unified Approach for Composing UML Aspect Models on Graph Transformation: Springer LNCS Vol. 5560, p. 191--237 (2009)

[38] Oliveira A., Seco N., Gomes P.: A CBR Approach to Text to Class Diagram Translation, In TCBR Workshop at the 8th European Conference on Case-Based Reasoning. (2004).

[39] Bajwa I., Samad A., Mumtaz S.: Object Oriented Software modeling Using NLP based Knowledge Extraction, European Journal of Scientific Research, 35(01), p.22--33 (2009)

[40] Kovacs L., Kovasznai G., Kusper G.: Metamodels in Generation of UML Using NLI-Based Dialogue. In 5th International Symposium on ACII, pp. 29--33 (2009)

[41] Bryant B., et al.: From Natural Language Requirements to Executable Models of Software Components. In Workshop on S. E. for Embedded Systems pp.51-58 (2008)