Enhancement Approachof Object Constraint Language Generation

Enhancement Approachof Object Constraint Language Generation

Samin Salemi and Ali Selamat

Faculty of Computing, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia

saminsalemi127@gmail.com

Abstract.OCL is the most prevalent language to document system constraints that are annotated in UML. Writing OCL specifications is not an easy task due to the complexity of the OCL syntax. Therefore, an approach to help and assist developers to write OCL specifications is needed. There are two approaches to do so: First, creating an OCL specifications by a tool called COPACABANA. Second, an MDA-based approach to help developers in writing OCL specification by another tool called NL2OCLviaSBVR that generates OCL specification automatically. This study presents another MDA-based approach called En2OCL, and its objective is twofold. 1- to improve the precison of the existing works. 2- to present a benchmark of these approaches. The benchmark shows that the accuracy of COPACABANA, NL2OCLviaSBVR, and En2OCL are 69.23, 84.64, and 88.40respectively.

1. Introduction

An OCL specification, is a Boolean expression that sets a condition on an entity, such as a class, attribute, data-type, and operation in UML models. It means that the condition must be true for all instances of a given entity. An OCL specification includes two main parts: context and expression body. The context of a OCL specification presents the entity restricted by the OCL specification, and the expression body of an OCL specification displays a Boolean condition. The entity, which is restricted by an OCL specification, is identified as a context variable of the OCL specification. Figure 1 presents the template of an OCL specification.

```
Context [contextVariable] [stereotype]:
[ExpressionBody]
```

Figure 1. OCL specification template

The complexity and difficulty of the OCL syntax causes some effects such as rising time and effort needed to create OCL specifications, and occurring errors while writing OCL specifications. The problems are motivations for embarking on this research. The main goal of the current research is to simplify the design phase of software modeling by proposing an MDA-based approach, to generate OCL specifications automatically. The existing tools for generating OCL specifications, use pattern-based and MDA-based approaches. The pattern-based approach can analyze the consistency of the constraint specifications automatically. This approach uses libraries to write OCL specifications. The

existing tool implementing the pattern-based approach is COPACABANA, which semi-automatically generates OCL specifications. The existing tool implementing the MDA-based approach is NL2OCLviaSBVR, which automatically generates OCL specifications by using Sitra.

2. Metamodels

Metamodel is the abstract syntax of a modeling language expressed as a model. The metamodel defines the structure of the model in terms of classes and relationships.

2.1. English Metamodel

In English metamodel, prefix element is a semantic element placed immediately before another semantic element. For example, in "valid customer card" sub-phrase, "valid" is a prefix element, because "valid" is an attribute and "customer card" is a class. Universal quantifiers are "all", "each", and "every". Existential quantifiers are "a", "an", "any", "s (plural), and "the". Negation element is "not". IsPropertyOfSign is a sub-phrase that links two semantic elements using "of". For example, "age of customer" is an IsPropertyOfSign. Valued range quantifier is a sub-phrase that determines a quantity by this syntax: "between quantity1 and quantity2" such as "between 3 and 5" and "between the number of customer cards and their owners". Possessive determiner is a sub-phrase of determiners that modify a noun by attributing possession to someone or something such as "customer's card". SignIntegratedWithAnd is a sub-phrase presenting a multiplication, division, addition, or subtraction of two quantitative things. For example, "the multiplication of the customer's age and the service's point" is a SignIntegratedWithAnd phrase. SumOf is a SignIntegratedWithAnd that adds two quantitative things. SubtractionOf is a SignIntegratedWithAnd that subtracts two quantitative things.MultiplicationOf is a SignIntegratedWithAnd that multiplies two quantitative things. DivisionOf is a SignIntegratedWithAnd that divides two quantitative things. Necessity verb is a modal verb showing a necessary action that must be performed. Transitive verb is a verb that takes one or more objects. Copular verb is a verb that links a subject to a complement that refers to the subject. Preposition conjunction is a preposition, such as "in" and "with", describing a relationship between two sub-phrases in a sentence. Str is a string. IsEqualTo can be "is", "are", "equals to", "equal to", "is equal to", or "are equal to".

2.2. SBVR Metamodel

SBVR is a standard to develop semantic models of business vocabularies and business rules [1]. SBVR concepts are object type and fact type. Object type is a general concept that is classified based on its characteristics. Fact type identifies a relationship among one or more object type. The object type in a fact type is called fact type role. Unary fact type (characteristic) has one fact type role, and binary fact type has two fact type roles. SBVR logical formulations are modal formulations, atomic formulation, logical operation, quantification, and objectification. Modal formulations used to formulate modality are divided into necessity and possibility. Necessity formulation is represented using the keywords "It is necessary" or "It is obligatory". Possibility formulation is represented using "It is possible" keyword. Atomic formulation specifies a fact type in a rule. Binary atomic formulation specifies a binary fact type in a rule. Quantification is a set of quantifications supported in SBVR and consists of universal quantification, at least n quantification, at most n quantification, etc. Objectification is a logical formulation that involves a bindable target. Logical operations are divided into logical negation and binary logical negation. Logical negation is a logical operation having one logical operand. Binary logical negation, such as conjunction, disjunction, and implication, is a logical operation having two logical operands.

2.3. OCL Metamodel

The context of an OCL specification is specified in the expression body using the self-keyword. An OCL context has a stereotype that can be one of these items: invariant, definition, initial, derivation,

pre and post-condition, and body. An OCL invariant expression specifies conditions on attributes and operations of a class in a Class diagram in form of arithmetic and logical operations. An attribute or association can be added to a Class diagram by OCL definitions. The initial value of attributes or associations can be specified by OCL initials. A derived value of an attribute or association end can be indicated by an OCL derivation. A pre-condition expression for an operation must be true before the operation execution and a post-condition expression for an operation must be true after the operation execution. OCL bodies are used to express query operation. The expression body of an OCL specification includes one or more of expressions such as let, if, literal, call, and unary operation. A let expression can define a new variable that has an initial value. An *if* expression defines a condition and two alternative expressions that after evaluating the condition, one of the alternative expressions is selected as the result of the if expression. A literal expression does not have any argument to produce a value. This kind of expression results in an expression symbol, such as an integer (12) and a string ("hello"). A call expression refers to an attribute, an operation, or an iterator for a collection, which is a collection of some values. An attribute call expression is a reference to a class's attribute in a UML model. A navigation call expression is a reference to a relationship in a UML model. An operation call expression is used, when we want to refer to an operation of a classifier. There are some unary operations, such as not Empty, is Empty, oclIsTypeOf, to perform functions on a value. The not Empty operation on a collection returns true when the collection has at least one element. The is Empty on a collection returns true when the collection has no element. The oclIsTypeOf operation on a value determines if a value is of the type given to the operation as a parameter. When we want to construct a loop over a collection, a loop expression is used. The for All operation takes an expression as a parameter and results to true if the expression is evaluated to true for all elements in the collection. The exists operation on a collection specifies a Boolean expression that must be true for at least one element of the collection. The select operation on a set, takes a parameter expression, and resulting a sub-set of the set, when the parameter expression is true for all elements of the resulting sub-set. The reject operation on a set takes a parameter expression and results to a sub-set of the set, when the parameter expression is false for all elements of the resulting sub-set. The collect operation on a collection gives the set of all values for a certain attribute of all objects in the collection.

3. Royal and Loyal Model

Warmer and Kleppe (1999) originally introduced the Royal & Loyal model [2]. Afterward, the model was used in various publications [3, 4, 5]. "*Royal and Loyal (R&L) models the computer system of a fictional company. It handles loyalty programs for companies that offer their customers various kinds of bonuses. Often, the extras take the form of bonus points or air miles, but other bonuses are possible as well: reduced rates, a larger rental car for the same price as a standard rental car, extra or better service on an airline, and so on. Anything a company is willing to offer can be a service rendered in a loyalty program*" [6]. Benchmark test cases which used in this research have been extracted from the Royal and Loyal model illustrated in Figure 2.

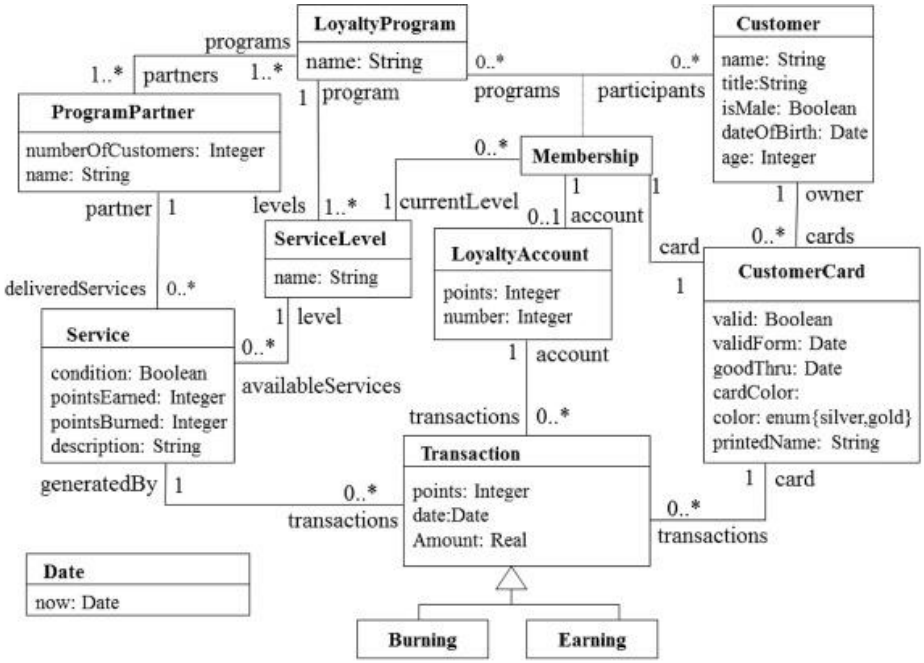

Figure 2. Royal and Loyal model [7]

4. En2OCL approach

The En2OCL approach proposed in this research, is based on the MDA approach to automatically transform system constraints formed in English sentences into OCL specifications. The proposed approach takes two inputs involving: an UML Class model, which presents a system and an English sentence, which presents a constraint of the system. The output of the proposed approach is a specification, which presents the system constraint in form of the OCL syntax. As Figure 3 illustrates, the approach contains three major analyses involving: lexical, syntactical, and semantic analysis involves.

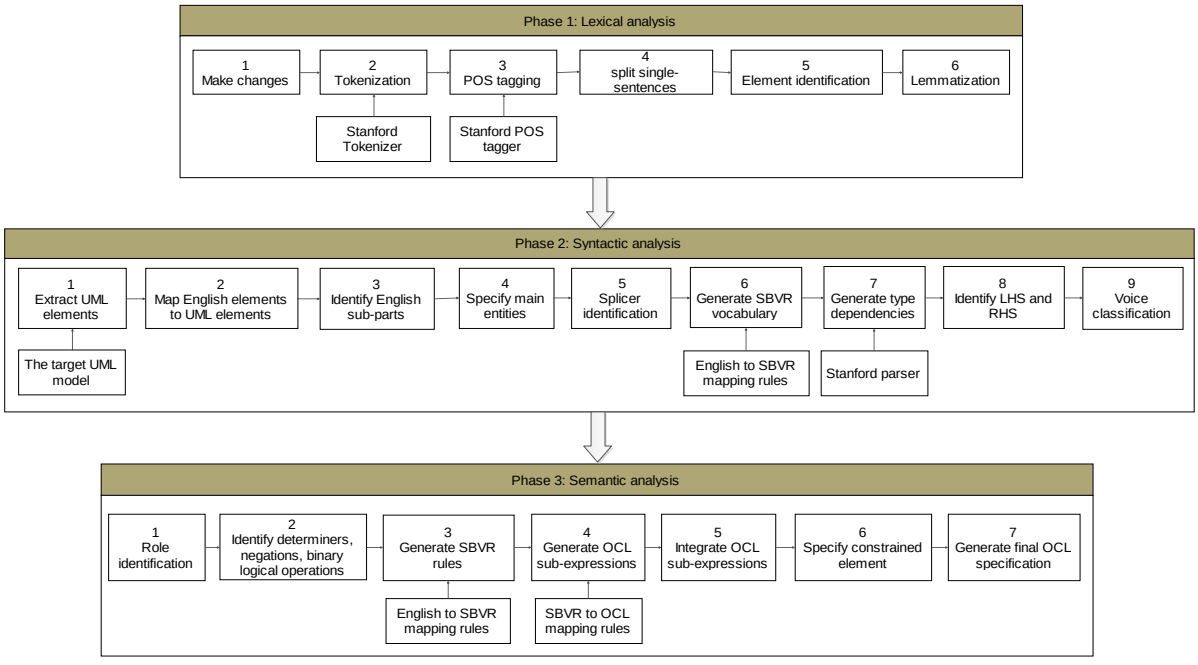

Figure 3. EN2OCL approach

The proposed approach, uses SBVR to bridge English elements to OCL elements. The Input English sentence is analyzed to extract business vocabulary and rules. The business vocabulary and rules are transformed into OCL specifications. Thus, two sets of mapping rules are created. The mapping rules that constitute the transformation are summarized in Table 1.

4.1. Lexical analysis

The lexical analysis includes six steps. In the first step, some parts of the input English sentence are changed. For example, “not more than” is changed to “at most”. In the two next steps, the changed sentence is tokenized and tagged by the Stanford Tokenizer and POS tagger. In the fourth step, the English sentence is split into single-sentences using the POS tags. In addition, duplicate single-sentences and single-sentences from which no business rules can be extracted are removed. In the fifth steps, nouns, adjectives, strings, and numeral elements are extracted from each single-sentence. In the sixth step, the extracted elements are lemmatized.

4.2. Syntactical analysis

The syntactical analysis of the input English sentence contains nine steps. In the first step, the elements of the UML Class model, such as classes, attributes, ends, and data-type are extracted. In the second step, the nouns and adjectives extracted from the English sentence are mapped to the elements extracted from the UML Class model. Strings, and numeral elements extracted from the lexical analysis, and then mapped elements are saved in an array. In the third step, English sub-parts, such as possessive determiners and prefix element, are identified. In the fourth step, main entities are selected. The main entities should not include data-type, possessive determiner, and prefix element. In the fifth step, the splicer between the main elements is specified. In the sixth step, the mapping rules are used to generate the SBVR vocabulary from the elements saved in the array and the splicers. In the seventh step, the type dependencies between the extracted elements are identified using the Stanford typed dependency parser. In the eighth step, the Left Hand Side (LHS), and Right Hand Side (RHS) elements are identified using the type dependencies. Left/right hand side element is the element placed in the left/right hand side of a verb or preposition conjunction. In the ninth step, it is determined that the splicer is active or passive. If the splicer is an active verb, the splicer has an active voice. However, if the splicer is a passive verb, the splicer has a passive voice.

4.3. Semantic analysis

The semantic analysis phase contains seven steps. In the first step, the role of characteristic fact type is specified. In addition, this step specifies the state of each binary fact type. In the second step, determiners, negation elements, and binary logical elements are extracted from each of the single-sentences. In the third step, SBVR rules are generated using the mapping rules. In this step, business rules are generated for the semantic formulations. In the fourth step, the business vocabulary and rules are translated to OCL sub-expressions using the mapping rules. In the fifth step, the sub-expressions are integrated. The sixth step determines which class of the UML Class model is being constrained. This Class is specified as the constrained element. In the seventh step, the final OCL specification is generated and revised.

Table 1. Mapping rules

Rule	English element	SBVR element	OCL element
Rule1	Necessity verb	Necessity	Constraint
Rule2	IsPropertyOfSign	Atomic formulation	AttributeCallExp
			NavigationCallExp
			Collect
Rule3	Possessive determiner	Atomic formulation	AttributeCallExp
			NavigationCallExp
			Collect
Rule4	Prefix element	Atomic formulation	AttributeCallExp
			Select
Rule5	Prefix element	Equivalence	EqualBool
			Select
Rule6	Transitive verb/ Copular verb	Atomic formulation	AttributeCallExp
			Collect
Rule7	Preposition conjunction	Atomic formulation	AttributeCallExp
			Collect
Rule8	Transitive verb/ Copular verb/ Preposition conjunction	Binary atomic formulation	Includes
			NavigationCallExp
			Collect
Rule9	Numeral	Number	NumericLiteralExp
Rule10	Noun (mapped to enumElement/enumName)	Objectification	Variable
Rule11	Noun (mapped to attribute/end)	Atomic Formulation	AttributeCallExp
			NavigationCallExp
Rule12	Noun (mapped to Class)	Object type	UMLElement(Class)
			OclIsTypeOf
			Select
			Size
Rule13	SignIntegratedWithAnd	Quantity	SignOpr
Rule14	SumOf	Quantity	Addition
Rule15	SubtractionOf	Quantity	Subtraction
Rule16	MultiplicationOf	Quantity	Multiply
Rule17	DivisionOf	Quantity	Division
Rule18	Sign	Quantity	RelationalOpr
Rule19	IsLessThan	Quantity	Less
Rule20	IsMoreThan	Quantity	More
Rule21	IsAtLeast	Quantity	AtLeast
Rule22	IsAtMost	Quantity	AtMost
Rule23	Str	Text	StringLiteralExp
Rule24	Existential quantifier (conditional)	Existential	notEmpty
Rule25	Universal quantifier	Universal	ForAll
Rule26	IsOrEqualOrEquals	Quantity	EqualNum
		Equivalence	EqualBool
		Equivalence	Select
		Quantity	Select
Rule27	AtMostN	AtMostN	AtMost
Rule28	AtMostOne	AtMostOne	AtMost
Rule29	AtLeastN	AtLeastN	AtLeast
Rule30	AtLeastOne	Existential	notEmpty
Rule31	MoreThanN	Quantity	More
Rule32	LessThanN	Quantity	Less
Rule33	ExactlyN	ExactlyN	EqualNum
Rule34	ExactlyOne	ExactlyOne	EqualNum

Rule35	Valued range quantifier	Numeric range	And
Rule36	Negation element	Logical negation	Not
			IsEmpty
			Excludes
Rule37	Binary logical operation	Binary logical Operation	LogicalOpr
Rule38	And	Conjunction	And
Rule39	Or	Disjunction	Or
Rule40	Conditional	Implication	Implies

5. Accuracy Benchmark Results

There are twenty six general test cases, which are used for benchmark of OCL generator tools. These twenty six benchmark test cases extracted from the Royal and Loyal case study have been applied on the three existing tools. The test cases and outputs of the three tools are presented below:

Constraint 1
English: *Every customer who enters a loyalty program must be of legal age.*
COPACABANA:context Customer inv legalAge: AttributeValueRestriction (Customer,age,>=,18)
NL2OCLviaSBVR: context Customer inv self.age >=
En2OCL: context loyaltyprogram inv: self.participants

In Constraint 1, the output of COPACABANA is complete and the outputs of NL2OCLviaSBVR and En2OCL are incomplete.

Constraint 2
English: *Male customers must be approached using the title "Mr".*
COPACABANA:-
NL2OCLviaSBVR: context Customer inv self.isMale implies self.title= Mr.
En2OCL: context customer inv: self->select(s|s.gender::male).title->exists(e|e="mr")

In Constraint 2, the output of COPACABANA is missed and the outputs of NL2OCLviaSBVR and En2OCL are complete.

Constraint 3
English: *The number of valid cards for every customer must be equal to the number of programs in which the customer participates.*
COPACABANA:context Customer inv maleTitle: IfThenElse ({ AttributeValueRestriction (Customer,isMale,=,true)}, AttributeValueRestriction (Customer,title ,=, 'Mr.'),)
NL2OCLviaSBVR: context Customer inv self.cards->select(valid=true)->size() =self.programs ->size()
En2OCL: context customer inv:self->forAll(a|a.programs->size()=a.cards-> select(s|s.valid)->size())

In Constraint 3, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.

Constraint 4
English: *The validFrom date of customer cards should be earlier than goodThru.*
COPACABANA:-
NL2OCLviaSBVR: context CustomerCard inv self.validFrom <self.goodThru
En2OCL: context customercard inv: self.validFrom <self.goodThru

---

In Constraint 4, the output of COPACABANA is missed and the outputs of NL2OCLviaSBVR and En2OCL are complete.

Constraint 5
English: *The birthdate of the owner of a customer card must not be in the future.*
COPACABANA:-

NL2OCLviaSBVR: context CustomerCard inv self.owner.dateOfBirth <> future
En2OCL: context customercard inv: self.owner.dateOfBirth.isBefore(Date::now)

In Constraint 5, the output of COPACABANA is missed and the outputs of NL2OCLviaSBVR and En2OCL are incomplete.
Constraint 6
English: *The owner of a customer card must participate in at least one loyalty program.*
COPACABANA:context CustomerCard inv programParticipation: MultiplicityRestriction
(CustomerCard,owner.programs,>,0)
NL2OCLviaSBVR: context CustomerCard inv self.owner.programs -> Size()>= 1
En2OCL: context customercard inv: self.owner.programs->notEmpty()

In Constraint 6, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 7
English: *There must be at least one transaction for a customer card with at least 100 points.*
COPACABANA:context CustomerCard inv transactionPoints: Exists (transactions,
{AttributeValueRestriction (CustomerCard,points,>,100)}
NL2OCLviaSBVR: context CustomerCard inv self.transaction->select(point >= 100)-> Size()>= 1
En2OCL: context customercard inv: self.transactions->exists(e|e.points>=100)

In Constraint 7, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 8
English: *The service level of each membership must be a service level known to the loyalty program.*
COPACABANA:context Membership inv knownServiceLevel: ObjectInCollection (Membership,
programs.levels, currentLevel)
NL2OCLviaSBVR: context Membership inv self.currentLevel ->includes (self.program.levels)
En2OCL: context loyaltyprogram inv: self.memberships->forAll (a|a.currentLevel =self.currentLevel)

In Constraint 8, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 9
English: *The participants of a membership must have the correct card belonging to this membership.*
COPACABANA:context Membership inv correctCard: bjectInCollection(Membership,
participants .cards, card)
NL2OCLviaSBVR: Context Membership inv self.participants.cards ->includes(self.card)
En2OCL: context membership inv: self.participants.cards ->includes(self.card)

In Constraint 9, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 10
English: *The color of a membership's card must match the service level of the membership.*
COPACABANA:context Membership inv levelAndColor1: IfThenElse({AttributeValueRestriction
(Membership, currentLevel.name,=,'Silver')}, AttributeValueRestriction
(Membership,card.color,=,Color::silver),) inv levelAndColor2:
IfThenElse({AttributeValueRestriction (Membership,
currentLevel.name,=,'Gold')}, AttributeValueRestriction
(Membership,card.color,=,Color::gold),)
NL2OCLviaSBVR: context Membership inv self.card.color = self.currentLevel.name
En2OCL: context membership inv: self.currentlevel.name = "silver" implies card.cardcolor =
color::silver and self.currentLevel.name = "gold" implies card.cardcolor = color::gold

---

In Constraint 10, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.

Constraint 11
English: *Memberships must not have associated accounts.*
COPACABANA:context Membership inv noAccount: MultiplicityRestriction
(Membership,account,=,0)
NL2OCLviaSBVR: context Membership inv self.account -> isEmpty()
En2OCL: context Membership inv self.account -> isEmpty()

In Constraint 11, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 12
English: *Loyalty programs must offer at least one service to their customers.*
COPACABANA:context LoyaltyProgram inv minServices: MultiplicityRestriction
(LoyaltyProgram,partners.deliveredServices,>=,1)
NL2OCLviaSBVR: context LoyaltyProgram inv self.partners.deliveredServices->size() >= 1
En2OCL: context loyaltyprogram inv: self.partners.deliveredservices->notEmpty()

In Constraint 12, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 13
English: *If none of the services offered in a loyalty program credits or debits the loyalty accounts, then these instances are useless and should not be present.*
COPACABANA:-
NL2OCLviaSBVR: context LoyaltyProgram inv
En2OCL: -

In Constraint 13, the outputs of COPACABANA and En2OCL are missed and the output of NL2OCLviaSBVR is incomplete
Constraint 14
English: *The name of the first level must be "Silver".*
COPACABANA:-
NL2OCLviaSBVR: context LoyaltyProgram invself.levels->first().name = Silver
En2OCL: context loyaltyprogram inv: self.levels->first ().name = “silver”

In Constraint 14, the output of COPACABANA is missed and the outputs of NL2OCLviaSBVR and En2OCL are complete
Constraint 15
English: *There must exist at least one service level with the “basic” name.*
COPACABANA:context LoyaltyProgram inv basicLevel: Exists(levels ,{ AttributeValueRestriction
(LoyaltyProgram,name,=,’basic’)})
NL2OCLviaSBVR: context ServiceLevel inv self.name = basic->exists(()
En2OCL: context servicelevel inv: self->exists(e|e.name->exists(e|e="basic"))

In Constraint 15, the outputs of COPACABANA and En2OCL are complete and the output of NL2OCLviaSBVR is incomplete
Constraint 16
English: *The number of participants in a loyalty program must be less than 10,000.*
COPACABANA:context LoyaltyProgram inv maxParticipants: MultiplicityRestriction
(LoyaltyProgram,participants,<,10000)
NL2OCLviaSBVR: context LoyaltyProgram inv: self.participants->size()<10000
En2OCL: context loyaltyprogram inv: self.participants->size()<10000

In Constraint 16, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.

Constraint 17
English: *The number of the loyalty account must be unique within a loyalty program.*
COPACABANA:-
NL2OCLviaSBVR: context LoyaltyProgram inv self.Membership.account->isUnique(acc|acc.numbers)
En2OCL: context loyaltyprogram inv: self.membership.account->size()=1

In Constraint 17, the output of COPACABANA is missed and the outputs of NL2OCLviaSBVR and En2OCL are complete.
Constraint 18
English: *The names of all customers of a loyalty program must be different.*
COPACABANA:context LoyaltyProgram inv uniqueNames:
UniqueSetIdentifier(LoyaltyProgram,participants,name)
NL2OCLviaSBVR: context LoyaltyProgram inv self.participants.name-
>forAll(c1,c2|c1.name<>c2.name)
En2OCL: context loyaltyprogram inv: self.participants.name->forAll(c1,c2|c1.name<>c2.name)

In Constraint 18, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 19
English: *The maximum age of participants in loyalty programs is 70.*
COPACABANA:context LoyaltyProgram inv maxAge: ForAll(participants,
{AttributeValueRestriction (LoyaltyProgram,age,<=,70)})
NL2OCLviaSBVR: context LoyaltyProgram inv: self.participants -> forAll(age<= 70)
En2OCL: context loyaltyprogram inv: self.participants.age<70

In Constraint 19, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 20
English: *There may be only one loyalty account that has a number lower than 10,000.*
COPACABANA:-
NL2OCLviaSBVR: context LoyaltyProgram inv self.Membership.account->one(number < 10,000)
En2OCL: context loyaltyaccount inv: self->one(o|o.number < 10000)

In Constraint 20, the output of COPACABANA is missed and the outputs of NL2OCLviaSBVR and En2OCL are complete.
Constraint 21
English: *The attribute numberOfCustomers of class programPartner must be equal to the number of customers who participate in one or more loyalty programs offered by this programPartner.*
COPACABANA:context ProgramPartner inv nrOfParticipants: MultiplicityRestriction
(ProgramPartner,programs.participants,=,numberOfCustomers)
NL2OCLviaSBVR: context ProgramPartner inv self.numberOfCustomers= programs.participants-
>asSet()->size()
En2OCL: context programpartner inv: self.numberofcustomers= self.programs.participants->size()

In Constraint 21, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.
Constraint 22
English: *A maximum of 10,000 pointsEarned may be earned using services of one partner.*
COPACABANA:-
NL2OCLviaSBVR: context ProgramPartner inv self.deliveredServices. pointsEarned<=10,000
En2OCL: context partner inv: self.deliveredservices.pointsEarned <=10000

In Constraint 22, the output of COPACABANA is missed and the outputs of NL2OCLviaSBVR and En2OCL are complete.

Constraint 23

English: *All cards that generate transactions on the loyalty account must have the same owner.*

COPACABANA:context LoyaltyAccount inv oneOwner: MultiplicityRestriction
(LoyaltyAccount,transactions.card.owner,=,1)

NL2OCLviaSBVR: context LoyaltyAccount inv self.transactions.cards.owner->asSet()->size() = 1

En2OCL: context loyaltyaccount inv: self.transactions.cards->forAll(a1,a2|a1.owner=a2.owner)

In Constraint 23, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.

Constraint 24

English: *If the points in a loyalty account is greater than zero, there exists a transaction with more than zero points.*

COPACABANA:context LoyaltyAccountinv positivePoints: IfThenElse({ AttributeValueRestriction
(points,>,0)}, Exists(transactions,{ AttributeValueRestriction
(LoyaltyAccount,points,>,0)}),)

NL2OCLviaSBVR: context LoyaltyAccount inv if (self.points > 0) then transaction -> exists(t|
t.points>0) endif

En2OCL: context loyaltyaccount inv: self.points>0 implies self.transaction->exists(e|e.points>0)

In Constraint 24, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.

Constraint 25

English: *There must be one transaction with exactly 500 points.*

COPACABANA:context LoyaltyAccount inv 500points: Exists(LoyaltyAccount,
transaction.points,{ LiteralOcl (−,"self = 500")})

NL2OCLviaSBVR: context Transaction inv self.transaction->select(point = 500)->Size()=1

En2OCL: context transaction inv: self->one(o|o.points=500)

In Constraint 25, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.

Constraint 26

English: *The available services for a service level must be offered by a partner of the loyalty program to which the service level belongs.*

COPACABANA:context ProgramPartner inv servicePartner: ObjectInCollection
(ProgramPartner,delivered Services.level .program.partners,Sequence{})

NL2OCLviaSBVR: context ServiceLevel inv self.program.partners->
includesAll(self.availableServices.partner)

En2OCL: context servicelevel inv: self.program.partners->includesAll (self.availableservices.partner)

In Constraint 26, the outputs of COPACABANA, NL2OCLviaSBVR, and En2OCL are complete.

COPACABANA, NL2OCLviaSBVR, and En2OCL have resulted in 18, 22, and 23 correct outputs, respectively. Thus, En2OCL had 19.23% accuracy improvement in comparison with COPACABANA and 3.84% accuracy improvement in comparison with NL2OCLviaSBVR. Figure 4 exhibits the accuracy of the tools.

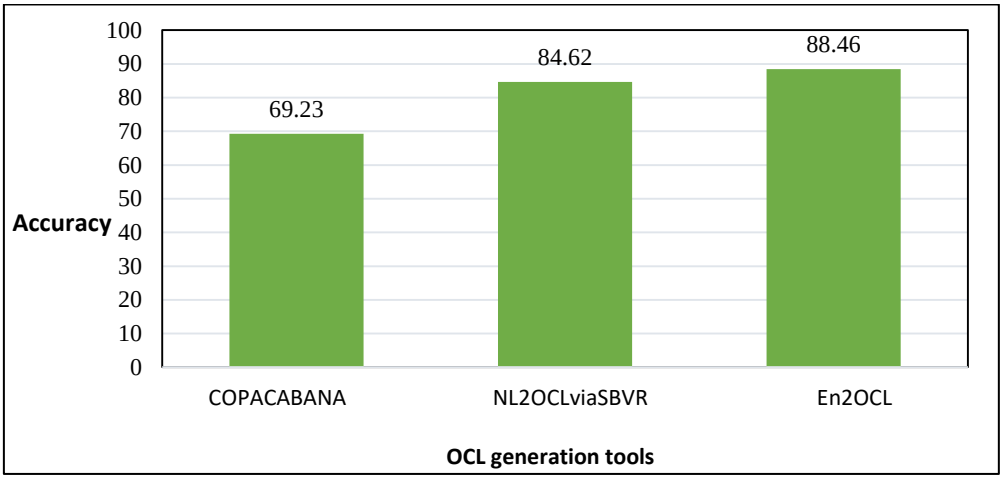

Figure 4. Accuracy benchmark results

6. Conclusion

The En2OCL approach proposed in this study is a transformation model to transform system constrains formed in English sentences into OCL specifications. The proposed approach overcomes English ambiguities using SBVR as an intermediate representation between English sentences and OCL specifications. En2OCL includes 21 major steps for generating an OCL specification from an English sentence. the proposed approach called En2OCL has been compared with the existing works (COPACABANA and NL2OCLviaSBVR). The accuracy comparison shows that the accuracy of En2OCL is 19.23% more than COPACABANA and 3.84% more than NL2OCLviaSBVR.

7. Reference

[1] OMG. (2013). Semantics of Business Vocabulary and Business Rules (SBVR), v1.2.
[2] Warmer, J., & Kleppe, A. (1999). Object Constraint Language: Precise Modeling with UML. Addison Wesley.
[3] Wahler, M. (2008). Using Patterns to Develop Consistent Design Constraints. ETH Zurich, Switzerland.
[4] Wilke, C., Thiele, M., & Wende, C. (2010). Extending Variability for OCL Interpretation. 13th International Conference on Model Driven Engineering Languages and Systems (MODELS 2010). October 3-8. Oslo, Norway.
[5] Bajwa, I. S., M. Lee, et al. (2012). "Translating natural language constraints to OCL." King Saud University–Computer and Information Sciences.
[6] Warmer, J. and A. Kleppe (2003). The Object Constraint Language: Getting Your Models Ready for MDA, Addison Wesley.
[7] Sharma, M., & Vishwakarma, R. G. Formalization & data abstraction during use case modeling in object oriented analysis & design. 3th International Conference on Computer Science, Engineering & Applications (ICCSEA 2013). May 24-26. Delhi, India: 67–75.