King Saud University
Journal of King Saud University – Computer and Information Sciences
www.ksu.edu.sa
www.sciencedirect.com

A model transformation framework to increase OCL usability

Samin Salemi [a,1], Ali Selamat [a,*], Marek Penhaker [b]

[a] UTM-IRDA Digital Media Centre and Faculty of Computing, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia
[b] Faculty of Electrical Engineering and Computer Science, VSB Technical University of Ostrava, Czech Republic

Received 28 June 2014; revised 23 March 2015; accepted 19 April 2015
Available online 2 November 2015

KEYWORDS

Model-Driven Architecture;
Model transformation;
OCL generation;
Usability improvement

Abstract The usability of a modeling language has a direct relationship with several factors of models constructed with the modeling language, such as time required and accuracy. Object Constraint Language (OCL) is the most prevalent language to document system constraints that are annotated in the Unified Modeling Language (UML). OCL is reputed as a modeling language with difficult syntax, and prior knowledge of OCL is needed to use the language. These obstacles result in the low usability of OCL. Therefore, the current research proposes a model to automatically transform system constraints formed in English sentences to OCL specifications. The proposed model is based on the Model-Driven Architecture (MDA) approach. The Linear Temporal Logic (LTL) properties of the proposed model are verified by the Maude model checker. To validate the proposed model and compare it with the existing work, the En2OCL (English2OCL) application is developed. This application is tested by three evaluation metrics: precision, recall, and f-measure. The En2OCL application is further compared with the NL2OCLviaSBVR application, which is the existing work on OCL generation from English sentences. The comparison shows a considerable improvement in precision, recall, and f-measure.

1. Introduction

Software designers use modeling languages to document system requirements and constraints. One of the significant characteristics of a modeling language is usability. There is a usability definition made by ISO: "*The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.*". As the ISO definition, usability is composed of three factors involving: effectiveness, efficiency, and satisfaction. Effectiveness is the ability of users to complete tasks using the system, and the quality of the output of those tasks. Efficiency is the level of resource consumed in performing tasks. Satisfaction is the subjective reaction of users to using the system.

OCL is the most prevalent modeling language to document system constraints that a software designer is not able to show by Unified Modeling Language (UML). The low usability of

* Corresponding author. Tel.: +60 7 5531008; fax: +60 7 5530160.
E-mail address: aselamat@utm.my (A. Selamat).
1 Tel.: +60 7 5531008; fax: +60 7 5530160.

Peer review under responsibility of King Saud University.

http://dx.doi.org/10.1016/j.jksuci.2015.04.002
1319-1578 © 2015 The Authors. Production and hosting by Elsevier B.V. on behalf of King Saud University.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

OCL due to its difficult syntax causes the gap between system constraints written in natural languages and OCL specifications to be bigger and bigger. Thus, the low usability of OCL increases time and effort spent on design phase of software development significantly then influences on overall development costs. In this research, a Model Driven Architecture (MDA)-based framework is proposed in order to increase OCL usability. The philosophy of MDA is to describe each artifact as a model and to transform the models to each other (Jilani and Usman, 2010). MDA provides a more efficient approach for software development, if the model transformations are done according to their specifications. This paper is structured as follows: in Section 2, two categories of the existing works are studied. The first category includes the existing MDA-based works, whose source models are a natural language. The second category includes the existing works for OCL generation. In Section 3, the contribution of the current study is presented. In Section 4, three metamodels for source, intermediate, and target models are presented. In Section 5, the proposed model is elaborated. In Section 6, the proposed model is evaluated.

2. Related works

There are some existing MDA-based works, whose source model is a natural language. For example, NIBA (Natural Language Based Requirements Analysis in German) presented by Fliedla et al. (2007) is an approach to analyze requirement descriptions linguistically and to translate them to a CS (Conceptual Schema). Amdouni et al. (2011) presented a tool to translate requirements formed in text documents to UML Class diagrams using NLP rules. Wang (2013) proposed EBD (Environment Based Design), which is a methodology to represent a natural language in conceptual models such as UML Use Case, Domain, and FBS (Function–Behavior–Stat e) models. NL2OCLviaSBVR developed by Bajwa (2012) is a tool, which is the only existing MDA-based work that supports OCL. The tool translates system constraints formed in English sentences to OCL specifications. On the other hand, there are some existing works to generate OCL specifications. For example, Wahler (2008) introduced an approach, which takes an unconstrained Class diagram and gives OCL specifications based on OCL patterns. The proposed approach analyzes the input Class models to elicit potentially missing constraints. There is a library of OCL constraint patterns, which allows developers to write OCL specifications according to the results elicited from the previous step. As OCL can be used as a model query language with a reputation as having a complex syntax, Störrle (2013) proposed an approach to identify the most important model query elements for ad hoc domain model querying. The model query elements are translated to OCL specifications in a library of query-predicates called OQAPI (OCL Query API). NL2OCLviaSBVR also can be categorized in OCL generators. However, the tool has some limitations. For example, it does not support some OCL elements such as collect, reject, enumeration, tuple data type, and XOR relations. The tool used SiTra (Simple Transformation), which has been developed by Akehurst et al. (2006) for implementing mapping rules. SiTra has some limitations, which are considered as the NL2OCLviaSBVR limitations. For example, one of the major limitations of SiTra regards a situation in which there is more than one rule that should map to the same target object. There is no way to determine, using SiTra, which of the rules should construct the target object.

2.1. Synthesis and evaluation of the related work

Some criteria are specified to evaluate the existing research works generating OCL specifications. These evaluation criteria are OCL application, input data, being pattern-based or not, level of automation, advantages, and limitations. OCL can query or constrain the state of a system. In other words, OCL can be used as a query language or as a constraint language. Thus, OCL has two applications involving: constraining and querying. The OCL application is an evaluation criterion that is considered in the existing works. The type of input data in the existing works is another evaluation criterion that is considered. OCL developers can write OCL specifications manually or can use OCL constraint patterns. An OCL constraint pattern is a library that allows developers to write OCL specifications based on the results elicited from the previous step. It must be specified if the OCL existing works are on the basis of patterns or not. Thus, pattern-based is another evaluation criterion. As there are different automation levels, automation level is another criterion for evaluation. Features, advantages, and drawbacks are some general evaluation criteria that are considered for the existing works. Table 1 presents the existing works evaluated using the criteria.

3. Contribution

Table 1 shows that there are only two existing works to generate OCL as constraint specifications. The first one is COPACABANA, which is a pattern-based tool and the second one is NL2OCLviaSBVR, which is an MDA-based tool. Table 2 presents some differences between the existing works and the proposed model. These differences show some limitations of the existing works solved by the proposed model.

4. Metamodels

4.1. English metamodel

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. Fig. 1 illustrates the classes of the English metamodel. The metamodel is created in the current research. `PossessiveDeterminer` is a sub-phrase of determiners that modify a noun by attributing possession to someone or something. For example, "*'s*" in "*customer's card*" is a possessive determiner. `UniversalQuantifier`, which expresses that some statements are true for everything or everybody, include "*all*", "*each*", and "*every*". `ExistentialQuantifier`, which expresses that some statements are true for something or somebody, include "*a*", "*an*", and "*s* (plural)". `TransitiveVerb` is a verb that takes one or more objects. `CopularVerb` is a verb that links a subject to a complement that refers to the subject. `NecessityVerb` is a modal verb showing a necessary action that must be performed. `IsEqualTo` includes "*is*", "*are*", "*equals*

Table 1 Evaluation summary of the existing works generating OCL specifications.

Researcher	Wahler (2008)	Störrle (2013)	Bajwa (2012)
OCL application	Constraint language	Model query language	Constraint language
Input	Unconstrained class model	English descriptions	UML Class model and English descriptions
pattern-based	Yes	No	No
Automation	Semi-automatic	Manual	Automatic
Tool/Library	COPACABANA	OQAPI	NL2OCLviaSBVR
Focus on usability	No	Yes	Yes
Feature work	Developing consistent constraint specifications based on constraint patterns	Identifying the most important model query elements for ad hoc domain model querying and translates them into OCL predicates	Converting natural language expressions to the equivalent OCL statements
Advantages	• Supporting users in detecting anti-patterns • Simplifying OCL constraint generation	• Improving the user-friendliness of OCL for ad hoc querying	• Simplifying the process of OCL statements generation • OCL usability improvement
Limitations	• Manual extraction of information from NL constraints • Manual selection of patterns • Low accuracy (69%) • Cannot using of return values of methods as parameter values in patterns • Required to substantial effort for adding new constraint patterns	• Not representative with any degree of certainty • Not easy to use for some people	• Limitations of SiTra • One input English sentence at a time • No support of collect • No support of reject • No support of enumeration • No support of tuple data-type • No support of XOR relations • No support of `OperationCall`

Table 2 Differences between En2OCL and the existing tools.

Property	COPACABANA	NL2OCLviaSBVR	En2OCL
Transformation language implementing mapping rules	–	SiTra	ATL
NavigationCall	No	Yes	Yes
Logical expression	Yes	Yes	Yes
Relational expression	Yes	Yes	Yes
Enumeration	Yes	No	Yes
includesAll()	No	Yes	Yes
Select()	No	Yes	Yes
XOR relations	No	No	Yes
isUnique()	No	Yes	Yes
Tuple data-type	No	No	No
collect()	Yes	No	Yes
reject()	Yes	No	Yes
`OperationCall`	Yes	No	No
Limited to process one English sentence at a time	–	Yes	No

to", "*equal to*", "*is equal to*", or "*are equal to*". `ValuedRangeQuantifier` 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*". `PrefixElement` is a semantic element placed immediately before another semantic element. For example, "*valid*" is a prefix element in "*valid customer card*", because "*valid*" is an attribute and "*customer card*" is a class. `PrepositionConjunction` is a preposition, such as "*in*" and "*with*", describing a relationship between two sub-phrases in a sentence. `NegationElement` is "*not*". `IsPropertyOfSign` is a sub-phrase that links two semantic elements using "*of*". For example, "*age of customer*" is `IsPropertyOfSign`. `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.

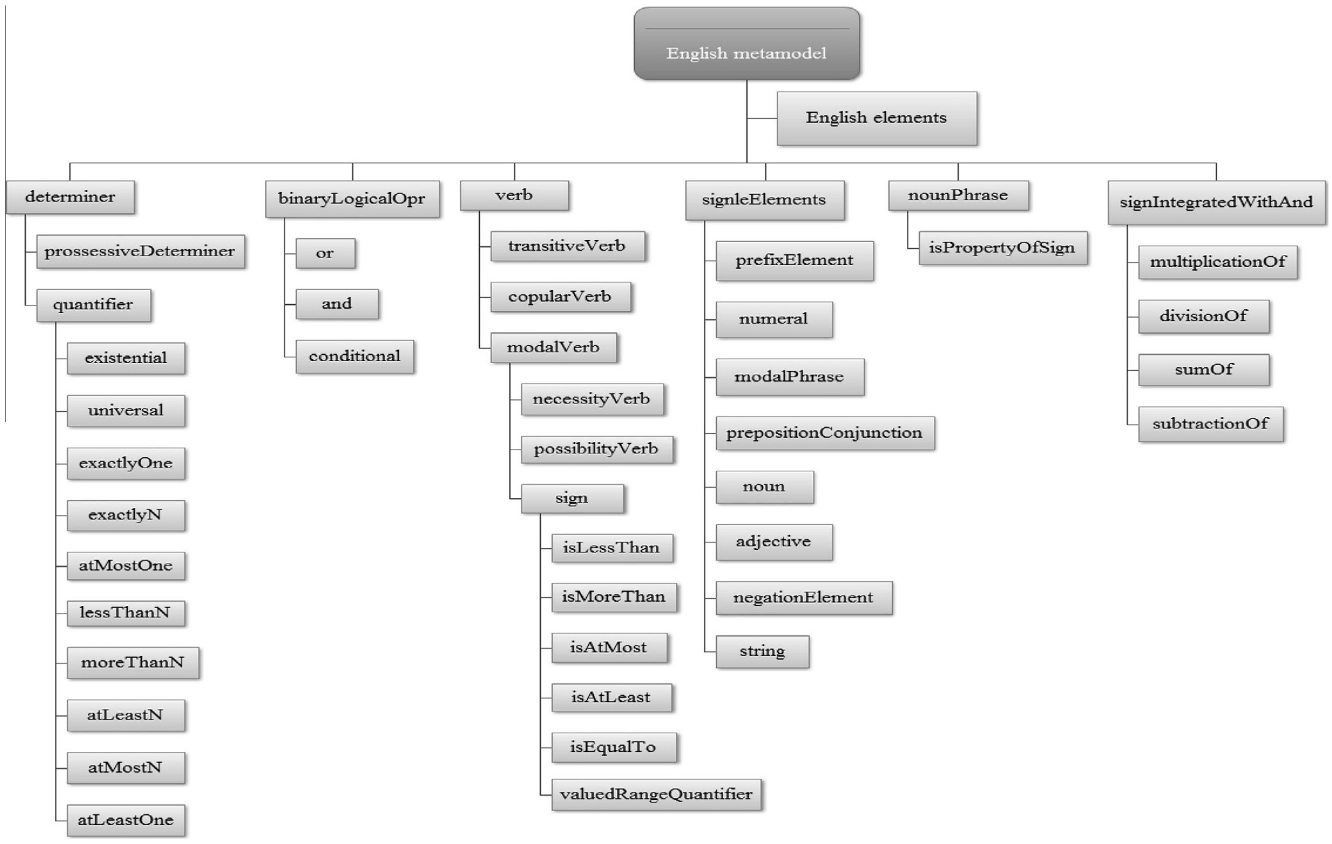

Figure 1 English metamodel.

4.2. Semantic Business Vocabulary and Rules

SBVR (Semantic Business Vocabulary and Rules) is a standard to develop semantic models of business vocabularies and business rules (OMG, 2013). The SBVR standard proposed by OMG is an integral part of MDA for formal description of a natural language (OMG, 2013). As natural languages such as English are informal descriptions, their translation to formal languages is so hard. The foundation of SBVR is on a formal logic (Bajwa, 2012), so translation of SBVR to other formal languages is easy. In the current research, SBVR is chosen as an intermediate representation. Thus, the elements of the English metamodel must be mapped into the elements of the SBVR metamodel and then the elements of the SBVR metamodel must be mapped into the elements of the OCL metamodel. SBVR provides business rules as a set of logical formulations to analyze semantics of natural languages. These business rules are written in natural languages. SBVR is a set of concepts and logical formulations (Njonko and El Abed, 2012). Formal vocabularies and rules produced by SBVR for a particular business domain can be used by computer systems. These vocabularies and rules help in robust semantic analysis of natural language texts. Concepts and Fact Types are two major elements of business vocabulary. A concept is a business entity in a particular domain and a fact type is a relationship between concepts in a business rule. SBVR proposes use of SBVR rules to represent particular business logic in a specific context. SBVR proposes a set of semantic formulations that semantically formulate the SBVR rules. Fig. 2 presents the classes of the SBVR metamodel. All the specific definitions of business concepts used by an organization or community are gathered in a SBVR business vocabulary. 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.

4.3. Object Constraint Language

Although, UML is the most common graphical modeling language, requirement constraints cannot be represented by it. Thus, a language, such as OCL that is the most common lan-

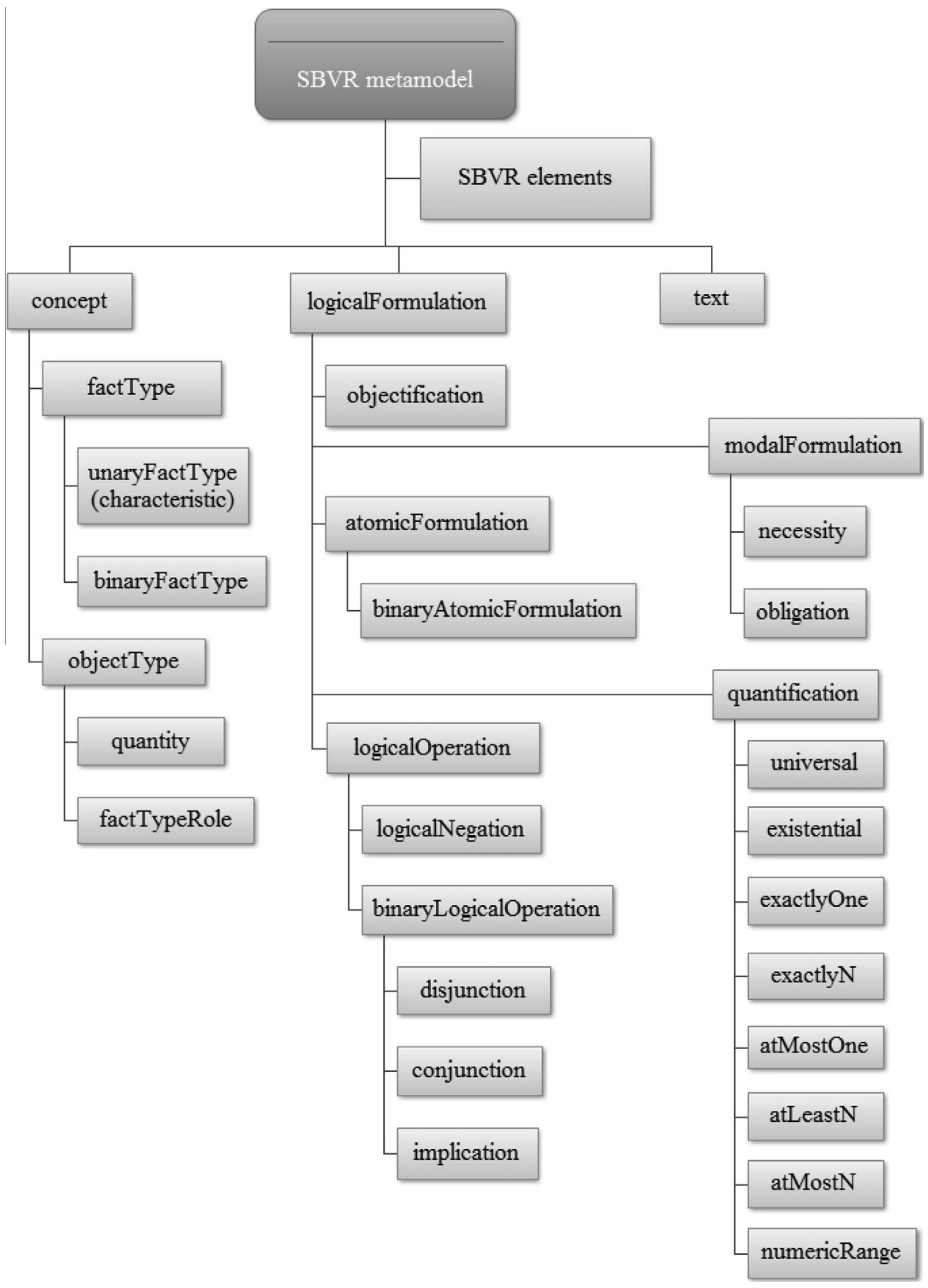

Figure 2 SBVR metamodel (OMG, 2013).

guage, is needed to document requirement constraints to improve the precision. In a system, requirement constraints specify how the system must operate or how it must be built. OCL specifications are annotated in UML. Model-Driven Engineering (MDE) is categorized OCL in the PIM level. 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 the entity. An OCL specification includes two main parts: context and expression body. The context of an 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. Fig. 3 shows the 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

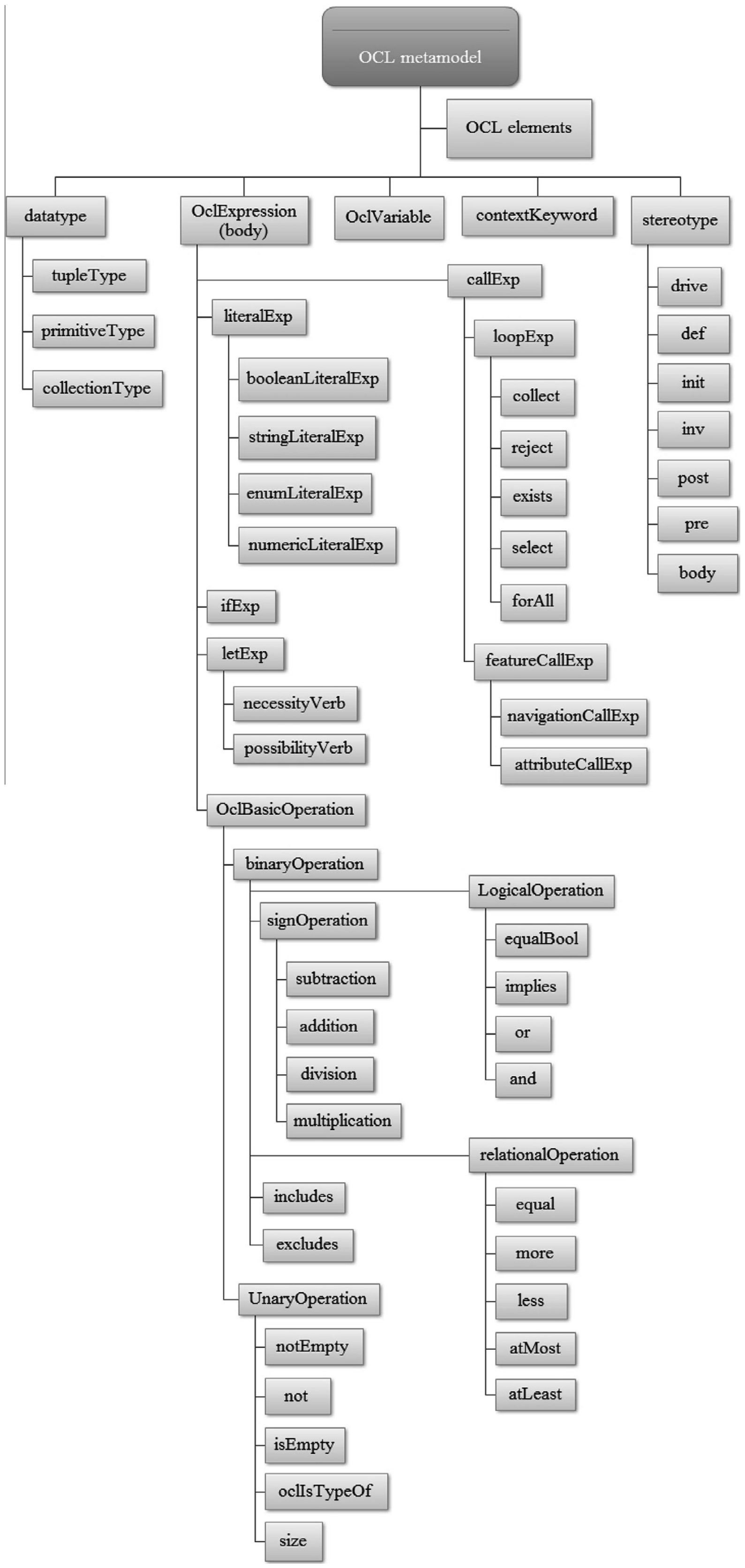

Figure 3 OCL metamodel (OMG, 2012).

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 notEmpty, isEmpty, oclIsTypeOf, to perform functions on a value. The notEmpty operation on a collection returns true when the collection has at least one element. The isEmpty 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 forAll 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 results to 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.

4.4. Royal and Loyal system as a case study

Warmer and Kleppe (1999) originally introduced the Royal & Loyal model. Afterward, the model was used in various publications. "*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*" Warmer and Kleppe (2003). The model of this system is illustrated in Fig. 4.

In the current study, the proposed model is implemented in an application. The application is demonstrated by applying it to the Royal and Loyal model as a case study. The case study examines how the application works. The strengths and weaknesses of the proposed model are identified using the case study. A set of test cases, which presents constraints of the famous model in form of English sentences, are provided then we try to transform the English sentences into OCL specifications.

5. Proposed model

The proposed MDA-based model automatically transforms system constraints formed in English sentences into OCL specifications. The model takes an English sentence and an UML Class model and gives an OCL specification. The proposed model uses SBVR to bridge English elements to OCL elements. The input English sentence is analyzed to extract English elements, which can be transformed to business vocabulary and rules. Then the business vocabulary and rules are translated to OCL expressions. Two set of mapping rules presented in Tables 3a–3c are generated for transforming English elements into SBVR elements and SBVR elements into OCL elements. This model presented in Fig. 5 contains three major analyses involving: lexical, syntactic, and semantic analysis.

5.1. Phase 1: Lexical analysis

The lexical analysis includes six steps. In the first step, some parts of the Input English sentence are changed. For example, modal phrases are removed and "*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 rule can be extracted are removed in this step. In the fifth next steps, nouns, adjectives, `ValuedRangeQuantifiers`, `SignIntegratedWithAnds`, strings, and numeral elements are extracted from each single-sentence. At the last step, the extracted elements are lemmatized. Lemmatization is a natural language processing task for grouping related elements and analyzing the related elements as a single item.

5.2. Phase 2: Syntactic analysis

The syntactic 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, dataType, and `enumElement`, are extracted. In the second step, the nouns and adjective extracted from the English sentence are mapped to the element extracted from the UML Class model. `ValuedRangeQuantifiers`, `SignIntegratedWithAnds`, strings, and numeral elements extracted from the lexical analysis and mapped elements are saved in an array named `elementArray`. In the third step, English sub-parts, such as `IsPropertyOfSign`, `PossessiveDeterminer`, and `PrefixElement`, are identified. In the fourth step, some elements are selected as main entities. If an element is not a datatype, right hand side of `IsPropertyOfSign`, left hand side of `PossessiveDeterminer`, and left hand side of `PrefixElement`, the element is a main entity. In the fifth step, the splicer between the main elements is

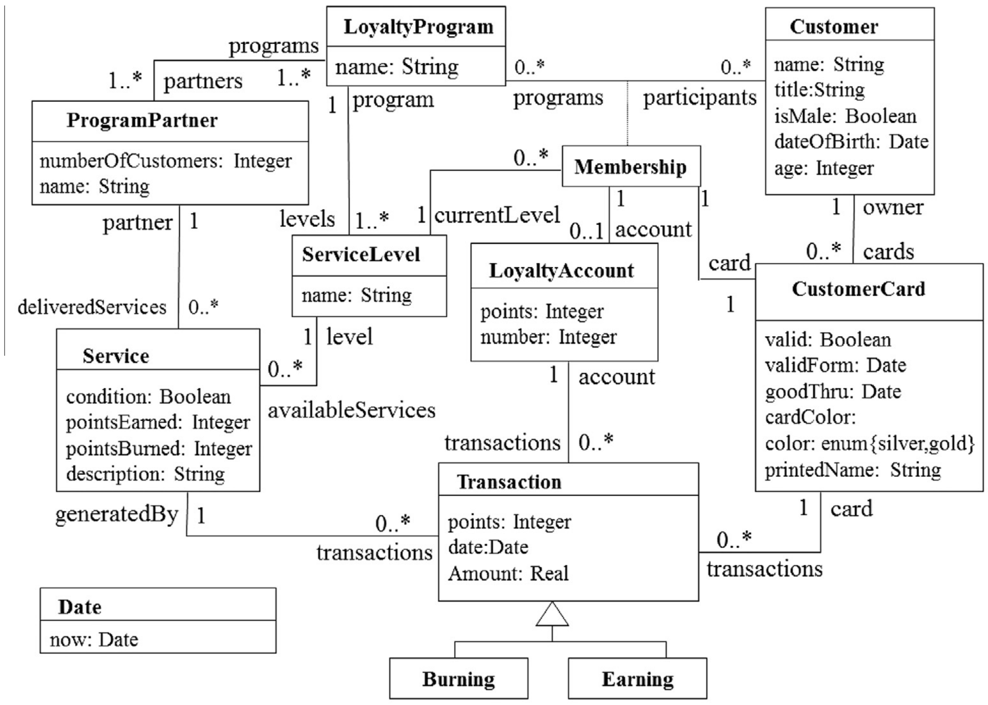

Figure 4 The Royal and Loyal model.

Table 3a Mapping rules.

Rule	English element	English example	SBVR element	OCL element	OCL example
Rule1	NecessityVerb	Must	Necessity	Constraint	context ... inv:
Rule2	IsPropertyOfSign	Name of customer Cards of customer Cards of customer	AtomicFormulation	AttributeCallExp NavigationCallExp Collect	customer.name customer.cards customer- > collect(c|c.cards)
Rule3	PossessiveDeterminer	Customer's name Customer's cards Customer's cards	AtomicFormulation	AttributeCallExp NavigationCallExp Collect	customer.name customer.cards customer- > collect(c|c.cards)
Rule4	PrefixElement	Valid customerCard Valid customerCard	AtomicFormulation	AttributeCallExp Select	customerCard.valid customerCard- > select(s| s.valid)
Rule5	PrefixElement	Silver cardColor Silver cardColor	Equivalence	EqualBool Select	cardColor =color::silver cardColor - > select(s|s =color::silver)
Rule6	TransitiveVerb/ CopularVerb	Customer has names Customer has names	AtomicFormulation	AttributeCallExp Collect	customer.name customer- > collect(c|c.name)
Rule7	PrepositionConjunction	Transaction with points Transaction with points	AtomicFormulation	AttributeCallExp Collect	transaction.point transaction- > collect(c|c.point)

specified. If a single-sentence has one main entity, there is no need to specify any splicer. But if a single-sentence has two main entities, the splicer between these two main entities is specified. In addition, it must be determined that the splicer between these two main entities is a relationship or a `restrictorRelationship`. Relationship splices two entities using a normal verb like "*each service has a service level*". `RestrictorRelationship` splices two entities using a

Table 3b Mapping rules.

Rule	English element	English example	SBVR element	OCL element	OCL example
Rule8	Numeral	28	Number	NumericLiteralExp	28
Rule9	Noun (mapped to enumElement/enumName)	Silver Color	Objectification	Variable	color::silver color
Rule10	Noun (mapped to attribute/ end)	Points Owner	Atomic Formulation	AttributeCallExp NavigationCallExp	transaction.points customerCard. owner
Rule11	Noun (mapped to Class)	Service Burning Burning Service	ObjectType	UMLElement (Class) OclIsTypeOf Select Size	service transaction.OclIsTypeOf(burning) transaction- > select(s|s.OclIsTypeOf(burning)) service- > size()
Rule12	SignIntegratedWithAnd	–	Quantity	SignOpr	–
Rule13	SumOf	The sum of the points of a loyaltyAccount and the loyaltyAccount's number	Quantity	Addition	loyaltyAccount. points + loyaltyAccount.number
Rule14	SubtractionOf	The subtraction of the points of a loyaltyAccount and the loyaltyAccount's number	Quantity	Subtraction	loyaltyAccount. points-loyaltyAccount.number
Rule15	MultiplicationOf	The multiplication of the points of a loyaltyAccount and the loyaltyAccount's number	Quantity	Multiply	loyaltyAccount.points*loyaltyAccount. number
Rule16	DivisionOf	The division of the points of a loyaltyAccount and the loyaltyAccount's number	Quantity	Division	loyaltyAccount.points/loyaltyAccount. number
Rule17	Sign	–	Quantity	RelationalOpr	–
Rule18	IsLessThan	Age is less than 28	Quantity	Less	age < 28
Rule19	IsMoreThan	Age is more than 28	Quantity	More	age > 28
Rule20	IsAtLeast	Age is at least 28	Quantity	AtLeast	age > = 28
Rule21	IsAtMost	Age is at most 28	Quantity	AtMost	age < = 28
Rule22	Str	'samin'	Text	StringLiteralExp	'samin'
Rule23	ExistentialQuantifier (conditional)	Customer has a name	Existential	notEmpty	customer.name- > notEmpty()
Rule24	UniversalQuantifier	All customerCard that is valid	Universal	ForAll	customerCard- > forAll(f|f.valid)
Rule25	IsOrEqualOrEquals	Age is 28 Name is 'Samin' Name is 'Samin' Age is 28	Quantity Equivalence Equivalence Quantity	EqualNum EqualBool Select Select	age = 28 name = 'Samin' name- > select(s|s = 'samin') age- > select(s|s = 28)
Rule26	AtMostN	loyaltyAccount has at most three points	AtMostN	AtMost	loyaltyAccount.points < = 3
Rule27	AtMostOne	loyaltyAccount has at most one points	AtMostOne	AtMost	loyaltyAccount.points < = 1
Rule28	AtLeastN	loyaltyAccount has at least three points	AtLeastN	AtLeast	loyaltyAccount.points > = 3

(*continued on next page*)

Table 3b (*continued*)

Rule	English element	English example	SBVR element	OCL element	OCL example
Rule29	`AtLeastOne`	`loyaltyAccount` has at least one points	Existential	`notEmpty`	`loyaltyAccount`.points->`notEmpty`()
Rule30	`MoreThanN`	`loyaltyAccount` has more than three points	Quantity	More	`loyaltyAccount`.points > 3
Rule31	`LessThanN`	`loyaltyAccount` has less than three points	Quantity	Less	`loyaltyAccount`.points < 3

Table 3c Mapping rules.

Rule	English element	English example	SBVR element	OCL element	OCL example
Rule32	`ExactlyN`	`loyaltyAccount` has exactly three points	`ExactlyN`	`EqualNum`	`loyaltyAccount`.points = 3
Rule33	`ExactlyOne`	`loyaltyAccount` has exactly one points	`ExactlyOne`	`EqualNum`	`loyaltyAccount`.points = 1
Rule34	`ValuedRangeQuantifier`	Point is between 3 and 5	`NumericRange`	And	points > 3 and points < 5
Rule35	`NegationElement`	Service1 is not service2 Memberships must not have any account The `customerCard` which does not include membership's cards	`LogicalNegation`	Not `IsEmpty` Excludes	not (service1 = service2) membership.account->`isEmpty`() `customerCard` -> excludes (memebrship.card)
Rule36	`BinaryLogicalOperation`	–	`BinaryLogicalOperation`	`LogicalOpr`	–
Rule37	And	A `loyaltyProgram` and customer can have a name	Conjunction	And	`loyaltyProgram`.name -> notEmpty() and customer.name -> notEmpty()
Rule38	Or	A `loyaltyProgram` or customer can have a name	Disjunction	Or	`loyaltyProgram`.nam-> notEmpty() or customer.name -> notEmpty()
Rule39	Conditional	If a customer's age is less than 18, the `customerCard` doesn't have any `customerCard`	Implication	Implies	customer.age < 18 implies customer. `customerCard` -> `isEmpty`()

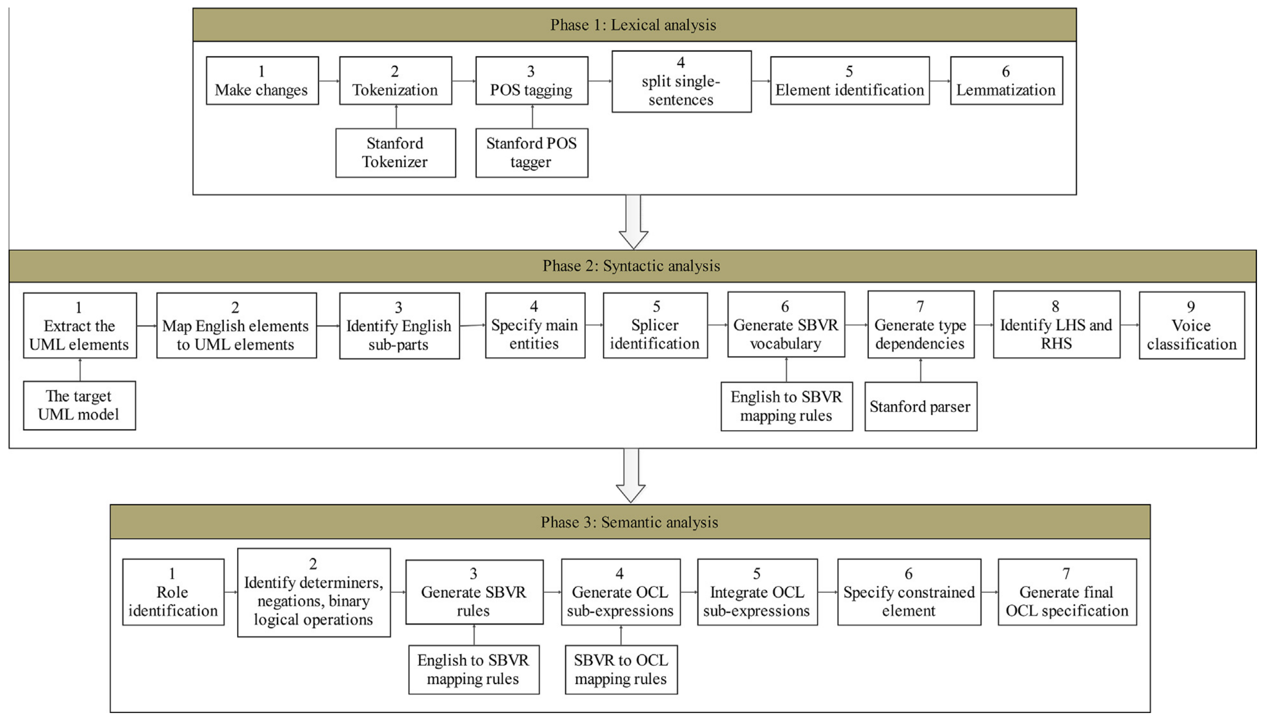
Phase 1: Lexical analysis
1 Make changes
2 Tokenization
3 POS tagging
4 split single-sentences
5 Element identification
6 Lemmatization
Stanford Tokenizer
Stanford POS tagger
Phase 2: Syntactic analysis
1 Extract the UML elements
2 Map English elements to UML elements
3 Identify English sub-parts
4 Specify main entities
5 Splicer identification
6 Generate SBVR vocabulary
7 Generate type dependencies
8 Identify LHS and RHS
9 Voice classification
The target UML model
English to SBVR mapping rules
Stanford parser
Phase 3: Semantic analysis
1 Role identification
2 Identify determiners, negations, binary logical operations
3 Generate SBVR rules
4 Generate OCL sub-expressions
5 Integrate OCL sub-expressions
6 Specify constrained element
7 Generate final OCL specification
English to SBVR mapping rules
SBVR to OCL mapping rules

Figure 5 En2OCL model.

coordinating conjunction (CC), preposition conjunction (IN), past participle verb (VBN), wh-determiner (WDT), wh-pronoun (WP), possessive wh-pronoun (WP$), or wh-adverb (WRB) like "*exactly one service which is owned by a program partner*". In the sixth step, the mapping rules presented in Tables 3a–3c are used to generate SBVR vocabulary from the elements saved in `elementArray` and the splicers. In the seventh step, the type dependencies between the extract elements are identified using the Stanford typed dependency parser. In the eighth step, LHS, and RHS elements are identifies using the type dependencies. Left/right hand side element is the element placed in the left/right hand side of a verb or `PrepositionConjunction`. In the ninth step, it is determined that the splicer is active or passive. If the splicer is an active verb or `PrepositionConjunction`, the splicer has an active voice. But if the splicer is a passive verb, the splicer has a passive voice.

5.3. Phase 3: Semantic analysis

The semantic analysis section contains seven steps. In the first step, role of characteristic fact type is specified. According to the Tables 3a–3c, `IsPropertyOfSigns` and `PossessiveDeterminers` are mapped to characteristic fact type. The role of the characteristic fact type mapped from an `IsPropertyOfSign` is the right element of the `IsPropertyOfSign`. The role of the characteristic fact type mapped from a `PossessiveDeterminer` is the left element of the `PossessiveDeterminer`. In addition, this step specifies the state of each binary fact type. There are three possible states for each binary fact type involving: a relationship without any related `restrictorRelationship` like "*each service has a servicelevel*", a relationship with a related `restrictorRelationship` like "*exactly one service which is owned by a program partner has a service level*", and a `restrictorRelationship` without any related relationship like "*There must be at least one transaction for a customer card*". In this step, the state of each binary fact type is specified. Role1 and Role2 of the binary fact types are specified. In the second step, determiners, negation, and binary logical elements of each single-sentence are identified. In the third step, the extracted elements are mapped to SBVR elements using mapping rules presented in Tables 3a–3c. As the Tables 3a–3c shows, existential determiners are mapped to existential quantification conditionally. The condition is this: "*a*", "*an*", "*the*", "*s* (plural)", "*any*" (when the splicer is a negated verb) are translated to existential quantification, if the object of the determiner is role2 of a binary fact type or the object of the determiner is role of an `unaryFactType`. In this step, business rules are generated for the semantic formulations involving: atomic formulations, quantifications, logical negations, and binary logical operations. In the fourth step, the business vocabulary and rules are translated to OCL sub-expressions using the mapping rules presented in Tables 3a–3c. In the fifth step, the OCL sub-expressions are integrated into one final expression. In the sixth step, it is determined that which class of the UML class model is being constrained using the final OCL expression. This class is specified as the constrained element and added to the final expression as its context element. The final expression and the context element is generate an OCL statement. In the seventh step, the OCL statement is revised.

6. Evaluation

Model validation checks the accuracy of the model's representation of a real system. To validate the model, a Java application called En2OCL is developed for representation of the

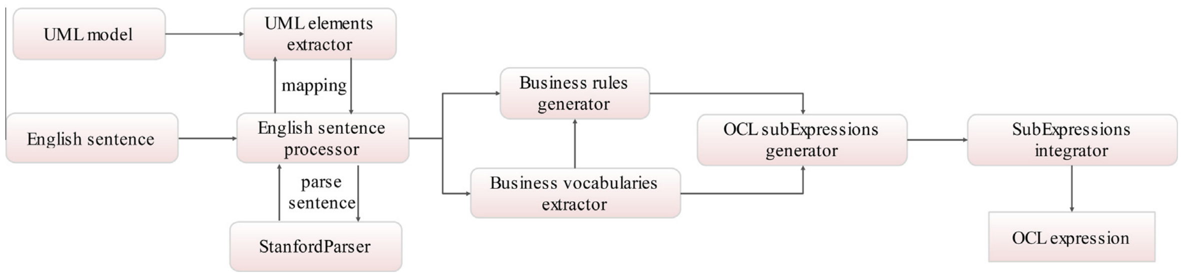

Figure 6 En2OCL application framework.

Table 4 Accuracy results.

Sentence type	Sample size	Correct outputs		Incorrect outputs		Accuracy	
		En2OCL	NL2OCLviaSBVR	En2OCL	NL2OCLviaSBVR	En2OCL	NL2OCLviaSBVR
Very simple	36	34	17	2	19	94.44	47.22
Simple	47	36	10	11	37	76.60	21.28
Medium	52	45	3	7	49	86.54	05.77
Complex	51	40	1	11	50	78.43	01.96
Total average	186	155	12	31	147	83.33	06.45

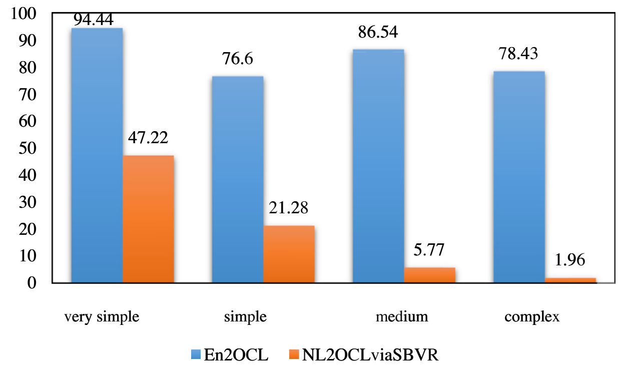

Figure 7 Accuracy comparison.

model in a real system. The En2OCL application takes two inputs: an English sentence and a UML Class model. The application then generates business vocabulary and rules from the English sentence using the first set of mapping rules. The application then generates an OCL specification from the business vocabulary and rules using the second mapping rules. Fig. 6 illustrates the application's framework. The input English sentences are the test cases extracted from the Royal and Loyal model. The test cases present constraints of the Royal and Loyal system. The English sentences are categorized in four groups involving: very simple, simple, medium, and complex.

6.1. Accuracy comparison

The accuracy of the application is measured and compared with NL2OCLviaSBVR, which is the only existing work generating OCL as constraint specifications from English sentences. These two applications are tested using 186 correct English sentences. Table 4 presents the test results. The comparison presented in Fig. 7 shows an improvement in the accuracy measure by En2OCL.

6.2. Usability measurement

ISO 9241-11 suggests that measures of usability should cover three factors involving: effectiveness, efficiency, and satisfaction. In this research, the efficiency factor is measured using effort-saving and time-saving items. The effectiveness factor is measured using writability and confidence items. The satisfaction factor is measured using ease-of-use and comfort items (Bajwa, 2012; Störrle 2013). These three usability factors are measured using a survey. We divided users into two categories: twenty medium users who have medium knowledge about OCL and twenty expert users who are expert in OCL. Ten English sentences and their corresponding OCL specifications were prepared to be used by the users. The users try to generate OCL statements in the three states: manually, using NL2OCLviaSBVR, and using En2OCL. Tables 5–7 present the users' responses in the three states. The comparison shows that there

Table 5 Users responses (manually).

User type	Sample size	Effort-saving	Time saving	Writability	Confidence	Ease-of-use	Comfort
Medium	20	2.45	1.95	2.80	2.80	2.40	1.55
Expert	20	2.95	2.85	2.10	2.50	3.65	3.45
Average		2.70	2.40	2.45	2.65	3.03	2.50

Table 6 Users responses (using NL2OCLviaSBVR).

User type	Sample size	Effort-saving	Time saving	Writability	Confidence	Ease-of-use	Comfort
Medium	20	3.75	3.95	3.00	3.20	4.10	4.25
Expert	20	3.65	3.70	2.80	3.00	4.10	4.10
Average		3.70	3.83	2.90	3.10	4.10	4.18

Table 7 Users responses (using En2OCL).

User type	Sample size	Effort-saving	Time saving	Writability	Confidence	Ease-of-use	Comfort
Medium	20	4.15	4.20	4.00	4.60	3.95	4.40
Expert	20	3.75	3.75	3.90	4.20	4.05	3.95
Average		3.95	3.98	3.95	4.40	4	4.18

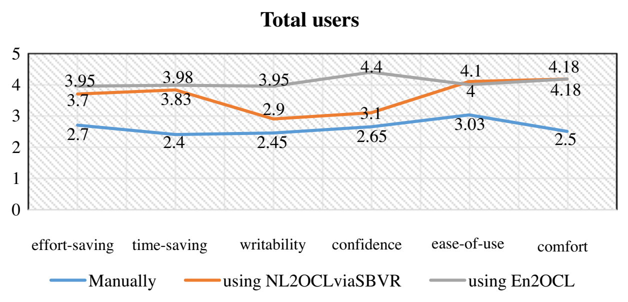

Figure 8: Usability comparison in total users

Figure 8 Usability comparison in total users.

are improvements in the six items using En2OCL than writing manually. The comparison also shows improvements in writability, confidence, effort-saving, and time-saving using En2OCL than usingNL2OCLviaSBVR. There is not any comfort improvement using En2OCL than using NL2OCLviaSBVR. Ease-of-use is reduced using En2OCL than using NL2OCLviaSBVR. Fig. 8 shows the usability comparison for total users.

7. Conclusion

It is common knowledge that OCL is reputed to be hard. The research shows that many modelers struggle with OCL, both in industry and academia, because prior knowledge of OCL is needed to use the language. These obstacles result in the low usability of OCL and thereafter the low adoption of OCL in industry. There are some existing MDA-based works, whose source model is a natural language. However, only one of these existing works supports OCL. On the other hand, there are a few existing works for OCL generation. Only two of the existing works generate OCL as constraint specifications. The first one is COPACABANA, which is a pattern-based OCL generator tool and the second one is NL2OCLviaSBVR, which is an MDA-based OCL generators tool. These two tools have some limitations presented in Table 2. Thus, this research proposed an MDA-based model for transforming system constraints formed in English sentences into OCL specifications automatically. For validating the proposed model, the proposed model was implemented in an application called En2OCL, and then En2OCL was compared with NL2OCLviaSBVR, which is the only existing MDA-based work on OCL generation from English sentences. The comparison showed there is an accuracy improvement by En2OCL. Furthermore, the measure of OCL usability is measured using a survey in three states involving: writing manually, using NL2OCLviaSBVR, and using En2OCL. The usability measurement showed there is a usability improvement by En2OCL than writing OCL specifications manually. Effectiveness and efficiency, which are two usability factors are improved by En2OCL than NL2OCLviaSBVR.

Acknowledgements

The Universiti Teknologi Malaysia (UTM) and Ministry of Education Malaysia under research university grants 00M19 and 01G72 are hereby acknowledged for some of the facilities that were utilized during the course of this research work. Also, this paper has been elaborated in the framework of the project "Support research and development in the Moravian-Silesian Region 2014 DT 1 - Research Teams" (RRC/07/2014). Financed from the budget of the Moravian-Silesian Region.

References

Akehurst, D.H., Boardbar, B., Evans, M., Howells, W.G.J., and McDonald-Maier, K.D., 2006. SiTra: simple transformations in Java. In: 9th International Conference on Model Driven Engineering Languages and Systems (LNCS), 351–364.

Amdouni, S., Karaa, W.B.A., and Bouabid, S., 2011. Semantic annotation of requirements for automatic UML Class diagram generation. The Computing Research Repository (CoRR). abs/1107.3297.

Bajwa, I.S., 2012. A Natural Language Processing Approach to Generate SBVR and OCL. University of Birmingham.

Fliedla, G., Kopa, C., Mayra, H.C., Salbrechtera, A., Vöhringera, J., Webera, G., et al, 2007. Deriving static and dynamic concepts from software requirements using sophisticated tagging. Data Knowled. Eng. 61 (3), 433–448.

Jilani, A.A.A., and Usman, M., 2010. Model transformations in model driven architecture: a survey. In: IEEE 2nd International Conference on Education Technology and Computer (ICETC), Shanghai, China.

Njonko, P.B.F., and El Abed, W., 2012. From natural language business requirements to executable models via SBVR. In: Proceeding of the International Conference on Systems and Informatics (ICSAI 2012). 19–20 May. Yantai 2453–2457.

OMG, 2012. OMG Object Constraint Language (OCL). OMG Document Number: formal/2012-01-01.

OMG, (2013). Semantics of Business Vocabulary and Business Rules (SBVR), v1.2.

Störrle, H., 2013. Improving the usability of OCL as an ad-hoc model querying language. In: 13th International Workshop on OCL, Model Constraint and Query Languages (OCL@MoDELS). CEUR-WS.org, United States, Miami, Florida, pp. 83–92.

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

Wang, M., 2013. Requirements Modeling: From Natural Language to Conceptual Models Using Recursive Object Model (ROM) Analysis. Concordia University, Montreal, Quebec, Canada.

Warmer, J., Kleppe, A., 1999. Object Constraint Language: Precise Modeling with UML. Addison Wesley.

Warmer, J., Kleppe, A., 2003. The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.