Automated transformation of NL to OCL constraints via SBVR

Murali Mohanan

Department of Computer Science,
College of Engineering,
Cherthala, Kerala, India
Email: mohananmurali@gmail.com

Abstract: This paper presents a neoteric method to automatically generate object constraint language (OCL) constraints from natural language (NL) statements. To support the software practitioners in using OCL, we present a novel method. The aim of this method is to produce a framework so that the user of unified modelling language (UML) tool can write constraints and pre/post conditions in natural language like English and the framework converts such natural language expressions to equivalent OCL statements. Here, the state of art of the two well-known technologies, namely open natural language processing (OpenNLP) and Semantics of Business Vocabulary and Rules (SBVR) are used. OpenNLP is used as a pre-processing phase to process the natural language statements. Pre-processing includes sentence splitting, tokenisation and parts of speech (POS) tagging. Then in the second phase, i.e., the transformation phase SBVR is used to automatically transform the preprocessed natural language statements to SBVR specifications. The main aim of this research is to provide automated tool support for model processing tasks in UML models via SBVR to model transform the input SBVR specifications to OCL specifications as explained in model driven architecture (MDA).

Keywords: natural language processing; Semantics of Business Vocabulary and Rules; SBVR; unified modelling language; UML; object constraint language; OCL.

Reference to this paper should be made as follows: Mohanan, M. (2020) 'Automated transformation of NL to OCL constraints via SBVR', *Int. J. Advanced Intelligence Paradigms*, Vol. 16, Nos. 3/4, pp.229–240.

Biographical notes: Murali Mohanan is an Associate Professor in the Department of Computer Science from the College of Engineering, Cherthala, Kerala, India. He is a Research Scholar in the Department of Computer Science, CUSAT, Kochi. He holds a BTech in Computer Technology and MTech in Computer and Information Sciences from the Cochin University of Science and Technology. He has two research publications in international conferences and journals. His research interests include NLP, UML modelling and software engineering.

This paper is a revised and expanded version of a paper entitled 'Automated transformation of NL to OCL constraints via SBVR' presented at 2nd International Conference on Inventive Research in Emerging Technology (ICIRET 2017), Bangkok, 30–31 January 2017.

1 Introduction

Due to rapid growth in information technology, the software engineering development has drastically changed and many innovative tools have emerged. Our tool is one such tool which automatically transforms natural language (NL) statements to Object Constraint Language (OCL). Open natural language processing (OpenNLP) is implemented to take care of the pre-processing of NL statements. The pre-processing phases are sentence splitting, tokenisation, parts of speech (POS) tagging and lemmatisation. Thus a controlled NL representation (Silvie and Healy, 2009) based on formal logic is used to address NL ambiguities and inconsistencies and make NL machine processable. The OpenNLP is used to produce the POS tags of user statements which is in the form of a NL such as English. The POS tags contain the required details such as noun, verb, adverb, etc., of the sentences.

In our approach, first phase is OpenNLP parsing and the second phase is Semantics of Business Vocabulary and Rules (SBVR) transformation. Here, SBVR vocabulary (such as concepts, fact types, etc.) is extracted. Then the SBVR vocabulary is further processed to construct a SBVR rule by applying SBVR's *conceptual formalisation and semantic formulation* (OMG Standard, 2008). The last phase is to apply the SBVR notation such as SBVR structured English or rule speak to formally generate SBVR rule statements. After SBVR vocabulary extraction and SBVR rules generation the statements are transformed to OCL constraints with the help of model transformation technology (Warmer and Kleppe, 2003; Ceponiene et al., 2015; Zikra et al., 2011).

2 Related works

Bajwa and Hyder (2007) present a NL processing based approach language engineering system for semantic analysis (LESSA) to understand the NL text and capture needed information automatically. 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 (Demuth and Wilke, 2009), IBM OCL Parser (IBM OCL Parser, 2009), USE (Ziemann and Gogolla, 2003) Cybernetic OCL Compiler, etc.

But these tools are limited to verify the syntax and type checking of the already written OCL code. During past years many automated solutions for transformation of NL software requirement specification (SRS) to unified modelling language (UML) based formal presentation have been presented (Ilieva and Olga, 2005; Jesús et al., 2009). Introduction of frameworks and tools for automated transformation NL to UML model have made things very easy and time saving for the software designers.

Cabot et al. (2010) also presented some transformation techniques to get semantically alike representations of an OCL constraint. The proposed technique assists in simplifying the modelling phase of software development by increasing the understanding level of the designer by providing him more than one alternate OCL representations.

Many related works have been done to transform OCL and UML to SBVR by Cabot et al. (2010). He proposed automatic transformation of UML and OCL schema to SBVR specification. This work is basically reverse engineering of software modelling 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. 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 milestone and it plays a major role in reducing gap between SBVR and UML based software modelling.

3 Preliminaries

In this sub section, SBVR is introduced.

3.1 Semantics of Business Vocabulary and Rules

The Object Management Group (OMG) introduced the SBVR (OMG Standard, 2008) in 2008 for software and business people. It describes the desired vocabulary and rules for providing the semantic documentation of vocabulary, facts and rules of business. It provides a multilingual, unambiguous and a rich capability of languages that are used by the people in various domains. SBVR offers a vocabulary for describing meaning of a sentence. The SBVR representation is simple to machine process as SBVR is based on formal logic (OMG Standard, 2008). SBVR can produce SBVR business vocabulary, SBVR business rules and SBVR business facts in a particular business domain.

3.2 Generating SBVR rules

In this phase, a SBVR representation such as SBVR rule is generated from the SBVR vocabulary in previous phase. The SBVR rule is based on any one of the fact type of SBVR vocabulary. Structural rule and behavioural rule are the two types of SBVR rules. The structural rule defines the organisational setup whereas behavioural rule describes the conduct of an entity. Semantic formulation, logical formulation, quantification and modal formulation are processes to be performed to generate the SBVR rules from the fact type. The SBVR rule is constructed by applying the semantic formulation to each fact type. Logical formulation, quantification, modal formulation and structured English notation are some of the formulations carried out.

3.2.1 Logical formulation

Using logical operators the multiple fact type is composed for SBVR rule. From the extracted vocabulary the required tokens are identified to map the logical operators. Tokens such as *not*, *no* are considered as the logical operator negation ($\neg$). *That*, *and* are denoted as conjunction ($\wedge$) similarly *or* is disjunction ($\vee$) and tokens like *infer*, *imply*, *indicate*, *suggest* are considered as the logical operator implication ($\rightarrow$).

3.2.2 Quantification

Quantification mentions the scope of the concept and it is applied in this work by mapping the tokens as given below:

More than, greater than → at least n quantification
Less than → at most n quantification
Equal to, positive statement → n quantification.

3.2.3 Modal formulation

The modal formulation describes seriousness of a constraint and it formulates modality. In SBVR two modal formulations are there, one is possibility formulation (PF) and the other is obligation formulation (OBF). The structural requirement is represented by the PF and the behavioural requirement is represented by the OBF. The model verbs mapping to these formulation is as shown below:

Can, may → PF
Verb concept, *should* → OBF.

3.2.4 Structured English notation used

The last step in SBVR generation is application of the structured English notation and the following formatting rules were used: The object types are underlined, e.g., *student*; the verb concepts are italicised, e.g., *should be*; the SBVR keywords are bolded e.g., *at most*; the individual concepts are double underlined, e.g., *Patron*. The characteristics are also italicised but with different colour: e.g., *name*.

From the generated SBVR rules, the informal mapping of SBVR and UML is carried out and it is as shown in Table 1.

Table 1 Informal mapping of SBVR and UML

	SBVR metamodel element	*UML metamodel element*
Informal mapping of SBVR and UML	Object type	Class
	Individual concept	Object
	Characteristic	Class attribute
	Verb concept	Class method
	Fact type	Association
	Partitive fact type	Generalisation
	Categorisation fact type	Aggregation
	Quantifications	Cardinalities

4 Object constraint language

OCL (OMG Standard, 2006) is a formal language which is easy to read and write. It is used to annotate a UML model with the constraints. In OCL, short and straight forward expressions are usually clear and easy to understand. Our paper is focused on usage of OCL constraints in representing functional requirements using class invariants, pre and post conditions on operations and other related expressions on a UML model (Ziemann and Gogolla, 2003; Gogolla et al., 2007; Linehan, 2006). By using suitable expressions,

OCL is suited for performing queries and checks on UML models. Non-functional requirements of a model can also be represented by OCL.

OCL expressions need to be defined in the context of a model which provides the information for validation and evaluation of OCL specifications. On the other hand, the information provided by textual specifications complements the information provided by the graphical formalism, UML. For instance, in UML diagram the graphical formalism reflects the fact that a company has at least one employee and a person may be employed by zero or more companies.

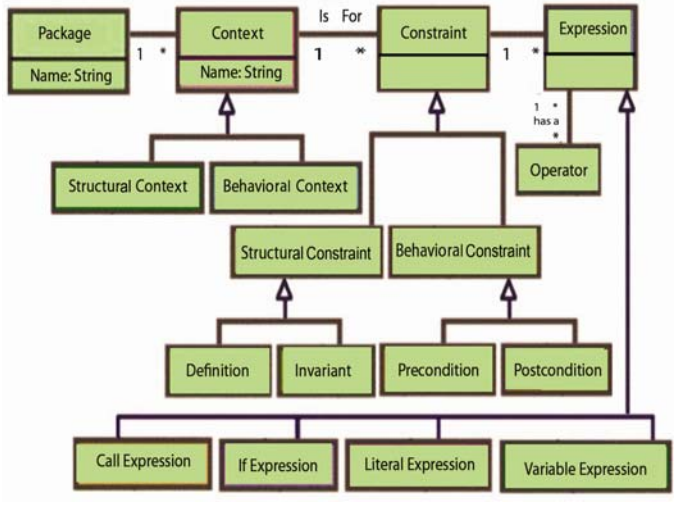
However, a person cannot be hired by a company unless he/she is at least 18 years old. This fact cannot be graphically illustrated, but can be specified in an OCL invariant for the person class:

context *Person* inv *major*:

self.company −> *notEmpty* () implies *self.age* >= 18

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. A selected set of OCL abstract syntax is used here for implementation. Figure 1 shows the abstract syntax metamodel (OMG Standard, 2006) of selected OCL expressions.

Figure 1 Elements of selected OCL meta-model (see online version for colours)



In the following paragraphs, important categories of OCL are discussed.

4.1 Model processing with OCL

OCL is a formal language primarily meant for expressing constraints in UML models. A constraint is a restriction on state or behaviour of an entity in a UML model (Ziemann and Gogolla, 2003; Jesús et al., 2009; Shah et al., 2009). Some of the important categories of OCL constraints are as follows:

4.1.1 Invariants

The invariants are conditions that have to be true for each instance of the method. Keyword used is 'inv'. Context employee, inv: self.age >= 18.

4.1.2 Precondition

A precondition is a constraint that should be true always before start of the execution of a method. Keyword used is 'pre'. Context Employee:: isAdult (dDOB: Integer): Boolean, pre: Ddob >= 1999.

4.1.3 Postcondition

It is also a constraint that should be true always after the execution of a method has finished. It describes what the state of a system is like after an operation and not how to modify the existing state to achieve the result. Keyword 'post' is used in post conditions. Context Employee:: isAdult (Ddob; Integer) : Boolean, post: result >= 18.

Considering the example of solving second degree equations we have the following OCL constraints as shown below:

```
ContextMath: solveSecondDegree (a: Real, b: Real, c: Real): Real
    Pre: b * b – 4 * a * c >= 0
    Post: a * result * result + b * result + c = 0
```

In the following section, generation of OCL constraints is explained.

5 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.

- extracting classes, methods and attributes from SBVR rules
- generation of OCL expressions from the extracted information, e.g., classes, methods and attributes
- mapping OCL syntax
- mapping OCL semantics.

5.1 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 in a UML model. The adjectives represent the attribute of a class or an instance.

5.2 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* person owns *at least one* account.

Step 1 The noun concept is mapped with the context.
i.e., customer or bank, e.g., context person

Step 2 Adding the instance.
i.e., bank person, e.g., person or self.

5.3 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 (OMG Standard, 2006).

Table 2 OCL syntax model

Syntax rule	*Syntactic elements*
Constraint	**Context** *Context-Name* **inv**:*Constraint-Body*
Context-Name	*Identifier* | *Identifier* :: *Identifier* [: *Class*] Context |
Identifier	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** | **all Instances** | **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 2. 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.

5.4 Mapping OCL semantics

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

Table 3 OCL operators for SBVR constructs

	Logical formulation	*OCL*	*Logical formulation*	*OCL*
OCL operators for SBVR constructs	Structural SBVR rule	a . b	b is *data type* of a	a:b
	Behavioural SBVR rule	a ● b	b is *return type* of a	a():b
	Method a of class b	a::b	Comments	– –

Structural or definitional business rules specify necessities on business concepts. Example is a rule that specifies how the total price of a rental is calculated. Behavioural or operational business rules specify business obligations and govern the conduct business activities. Example, a rule stating, 'blacklisted customers must not be given a rental' is an operative business rule.

The SBVR rule is transformed to OCL constraint using transformation rules as given below:

Step 1 *It is necessary that* a person owns *at least one* account.

Step 2 *It is necessary that* <Quantification><actor>owns<Quantification><attribute>.

Transformation rules are then applied to generate the OCL constraint. A typical model transformation rule comprises of the variables, predicates, queries, etc (Silvie and Keri, 2009). A transformation rule consists of two parts: a left-hand side (LHS) and a right-hand side (RHS). 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 specified in the following rules. Rule 1 transforms the *noun concept* (actor) in SBVR rule to OCL context for an invariant and 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*.

Rule2 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 rule 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*)]

- contextcontext-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.

The above mentioned rules were used for SBVR to OCL transformation. We have properly addressed the invariants and pre/post conditions. We have designed generic templates for common OCL expressions such as OCL invariant, OCL pre-condition, and OCL post-condition. User has to select one of these three templates manually. Once the user selects one of the constraints, the missed elements in the template are extracted from the logical representation of English constraint. Following are some of the templates used for the work.

1 The template used for invariant is:

package [UML-Package]

context [UML-Class]

inv: [Body]

2 The template used for OCL pre-condition is:

package [UML-Package]

context [UML-Class::Class-Op(Param):Return-Type]

pre: [Body]

3 The template used for OCL post-condition is:

package [UML-Package]

context [UML-Class::Class-Op(Param):Return-Type]

post: [Body]

Result: [Body] – optional

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

1 UML-Package is the package name of the target UML class model.

2 UML-Class is the name of the class in the target UML Class model and UML-Class should also be an Object Type in the subject part of the English Constraint.

3 Class-Op is one of the operations of the target class (such as context) in the UML class model and Class-Op should also be the Verb Concept in English constraint.

4 Param is the list of input parameters of the Class-Op and we get them from the UML class model. These parameters should be of type Characteristics in English constraint.

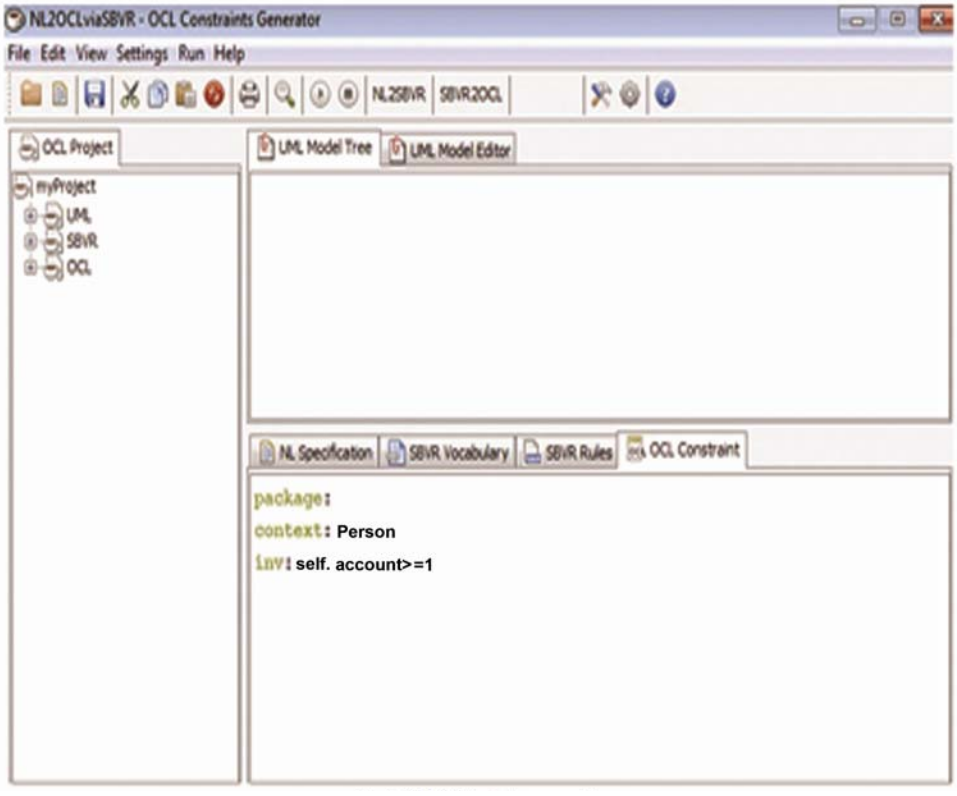
5 Return-Type is the return data type of the Class-Op and we get them from the UML class model. The return type is the data type of the used Characteristic in English constraint and this data type is extracted from the UML class model.

Body can be a single expression or combination of more than one expression.

6 Tool implemented

Successfully implemented a prototype tool to translate NL text to OCL constraints with the support of SBVR. SBVR transformation rules and the abstract syntax of OCL generations were implemented in NetBeans platform using Java. An OpenNLP parser is implemented to syntactically analyse the NL statements and then to map with SBVR rules. Finally, SBVR specification to OBL translation is carried out by using translation rules. A screen shot of the developed tool is as shown in the Figure 2. We have generated a sample OCL output for the simple SBVR statement, i.e., It is necessary that each person has at most one account and is shown in the screen shot.

Figure 2 NL2OCL tool – a screenshot (see online version for colours)



7 Experiments and results

To test the accuracy of OCL constraints various complexity levels of input, i.e., simple, compound and complex SBVR rules are defined and are as given below:

- Simple: *It is necessary that each* customer has *at most one* bank account.
- Complex: *It is obligatory that each* customer can have *at least one* bank account *only if* age of customer is *18* years.
- Compound: *It is permitted that if each* customer has *at least* one bank account and account balance is *at least* $1,000 can apply for *at least* one credit card.

Various examples for the different complexity levels are tested and the results are tabulated as shown in Table 4.

Table 4 OCL constraints test results

Constraint type	*Invariant*	*Precondition*	*Postcondition*	*Total*
Simple	93.5%	90.5%	88.1%	90.7%
Complex	90.1%	89.2%	85.6%	88.3%
Compound	83.5%	81.5%	75.4%	80.1%

The overall accuracy is calculated to 86.37% and is very encouraging.

8 Conclusions

Our research paper aims to provide an automated tool support for model processing tasks in UML models. This is achieved by extracting Object Oriented items from user provided NL specifications with the help of SBVR and then simultaneously generating OCL constraints from that. The NL processed input is extracted to find out the noun concepts, individual concepts, verbs and adjectives to constitute a complete SBVR rule. The SBVR rules are then 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. Presently, this implemented prototype tool can process one sentence at a time and this can be extended to process multiple constraints and integrated into existing tools.

References

Bajwa, I.S. and Hyder, I. (2007) 'UCD-generator – a LESSA application for use case design', *Proceedings of IEEE-International Conference on Information and Emerging Technologies – ICIET*, pp.182–187.

Cabot, J., Pau, R. and Raventós, R. (2010) 'From UML/OCL to SBVR specifications: a challenging transformation', *Information Systems,* Vol. 35, No. 4, pp.417–440.

Ceponiene, L., Nemuraite, L. and Vedrickas, G. (2015) 'Separation of event and constraint rules in UML&OCL models of service oriented information systems', *Information Technology and Control*, Vol. 38, No. 1.

Demuth, B. and Wilke, C. (2009) 'Model and object verification by using Dresden OCL', in *R.G. Workshop on Innovation Information Technologies: Theory and Practice*, pp.81–89.

Gogolla, M. et al. (2007) 'USE: a UML-based specification environment for validating UML and OCL', *Science of Computer Programming*, Vol. 69, Nos. 1–3, pp.27–34.

IBM OCL Parser (2009) [online] http://www-01.ibm.com/ software/awdtools/library/standards/ocl-download.html (accessed 2016).

Ilieva, M. and Olga, O. (2005) *Automatic Transition of Natural Language Software requirements Specification into Formal Presentation*, Vol. 3513, pp.392–397, Springer LNCS, Alicante, Spain.

Jesús, S.C., Frédéric, J., Jesús, G.M. and Jean, B. (2009) *Optimization Patterns for OCL-Based Model Transformations*, Vol. 5421, pp.273–284, Models in Software Engineering LNCS, Springer, Berlin, Heidelberg.

Linehan, M. (2006) 'Semantics in model driven business design', in *2nd International Conference on Semantic Web Policy Workshop*, Athens, GA, USA, pp.1–8.

OMG Standard (2006) *OMG: Object Constraint Language (OCL)*, v. 2.0.

OMG Standard (2008) *OMG: Semantics of Business Vocabulary and Rules (SBVR)*, v. 1.0.

Shah, A., Anastasakis, K. and Bordbar, B. (2009) 'From UML to alloy and back', in *ACM International Conference Proceeding Series*, Vol. 413, pp.1–10.

Silvie, S. and Keri, A. (2009) 'SBVR's approach to controlled natural language', *Workshop on Controlled Natural Language*, Marettimo Island, Italy, June, pp.8–10.

Spreeuwenberg, S. and Healy, K.A. (2009) 'SBVR's approach to controlled natural language', *International Workshop on Controlled Natural Language*, Springer, Berlin, Heidelberg, pp.155–169.

Warmer, J.B. and Kleppe, A.G. (2003) *The Object Constraint Language: Getting Your Models Ready for MDA*, Addison-Wesley Professional, Boston, MA.

Ziemann, P. and Gogolla, M. (2003) 'Validating OCL specifications with the use tool: an example based on the Bart case study', *Electronic Notes in Theoretical Computer Science*, Vol. 80, pp.157–169, Roros, Norway.

Zikra, I., Stirna, J. and Zdravkovic, J. (2011) 'Analyzing the integration between requirements and models in model driven development', *Enterprise, Business-Process and Information Systems Modeling*, pp.342–356, Springer, Berlin, Heidelberg.