OCL Usability: A Major Challenge in Adopting UML

Imran Sarwar Bajwa
Dept. of Computer Science & IT
Islamia University of Bahawalpur
63100, Pakistan
+92 (0)62 925 5466
imran.sarwar@iub.edu.pk

Behzad Bordbar
School of Computer Science
University of Birmingham
B15 2TT, UK
+44 (0)121 414 3487
b.bordbar@cs.bham.ac.uk

Mark Lee
School of Computer Science
University of Birmingham
B15 2TT, UK
+44 (0)121 414 4765
m.g.lee@cs.bham.ac.uk

ABSTRACT
In this paper, we present a novel approach to address the OCL usability problem by automatically producing OCL from English text. The main aspects of OCL usability problem are attributed as hard syntax of language, ambiguous nature of OCL expressions, and difficult interpretation of large OCL expressions. Our contribution is a novel approach that aims to present a method involving using Natural Language expressions and Model Transformation technology to improve OCL usability. The aim of the method is to produce a framework so that the user of UML tools can write constraints and pre/post conditions in English and the framework converts such English expressions to the equivalent OCL statements. The proposed approach is implemented in a software tool NL2OCLviaSBVR that generates OCL constraints from English text via SBVR. Our tool allows software modelers and developers to generate well-formed OCL expressions that results in valid and precise models. An empirical evaluation of the OCL constraints reveals that our natural language based approach to generate OCL constraints significantly outperforms the most closely related technique in terms of effort and effectiveness.

Categories and Subject Descriptors
D.3.3 [**Programming Languages**]: Language Constructs and Features – *abstract data types, polymorphism, control structures.*

General Terms
Theory, Design, Experimentation, Languages, Verification

Keywords
OCL, Constraints, Natural Language Processing

1. INTRODUCTION
The Unified Modeling Language (UML) [1] is now widely considered as a de facto standard family of languages for specifying, modelling, constructing and documenting object-oriented software and systems. Popularity of UML is often attributed to its semi-formal nature. It is argued that UML is not formal enough to demand deep knowledge of formal methods that inhibits practically minded software engineers from using it. As a result, usability is seen as a major feature of the UML. However, Object Constraint Language (OCL) [2], which is one of the languages in UML, is a clear exception to this argument. OCL plays a key role in UML modelling for expressing essential constraints to make UML models well-defined. But it is also a common knowledge that OCL is the least adopted amongst all languages in UML [4].

We have identified three major factors contributing to usability problems in OCL. The primary factor is the hard syntax of OCL [3]. Wahler [4] addressed this problem by introducing a template based language. His approach, which is implemented to be used with IBM Rational, allows the user to pick a template, from a wide range of OCL template, assign the parameters and use them. This would greatly help the user; however the key challenge is to learn which template to pick. Second aspect of OCL's usability problem is the ambiguous nature of OCL constraint as several equivalent implementations for a constraint are possible in OCL [5, 21]. Cabot proposed an approach for automatic disambiguation of the constraints by means of providing a default interpretation for each kind of ambiguous expression. But a designer has to be aware of all the possible states while writing an OCL constraint to avoid the identified ambiguities. Third aspect of OCL's usability problem is understandability of overly complex OCL expressions commonly used in large software models [6]. The refactoring techniques are used to improve the understandability of OCL specifications but the employment of refactoring technique can be an overhead in the process of software modelling.

To contribute to OCL's usability a tool has to be able to deal with English. In practice, an English expression of a constraint is manually mapped to an OCL constraint on a given UML model. We have identified a set of tasks that are involved in typical English to OCL mapping. Firstly, since OCL is side-effect free [2], the English statement must be about the system, i.e. the terms and vocabulary used must be already existence in the model. Secondly, English language is inherently ambiguous. It is important to start from a correct understanding of the meaning of the expression. Thirdly, we believe that if the English sentence is clear and well-understood, the creation of OCL can be automated. On the basis of the above three points, this paper presents a framework for automated creation of OCL statements from the English language expressions.

Conference'10, Month 1–2, 2010, City, State, Country.

Our approach allows the user to write various constraints and pre/post conditions on a UML model in English. First input the English is mapped with input UML model. Then the English constraints are automatically transformed to the equivalent OCL expressions via SBVR [7] (Semantic Business Vocabulary and Rules). SBVR is an OMG's recent standard that and we have used SBVR to overcome the inherent ambiguity of English language. SBVR not only provides English a semantically formal representation but also closed to OCL syntax as both languages are based on formal logic. To create an OCL expression, the SBVR rules are transformed to OCL using MDA model transformations. As a proof of concept, the proposed approach is implemented as an Eclipse plugin called NL2OCLviaSBVR. The NL2OCLviaSBVR automatically transforms English to OCL via SBVR. The automated transformation not only hides the complexity involved in the manual production of OCL constraints from English language but also results in producing OCL constraints in a seamless and non-intrusive manner.

The rest of the paper is structured as follows: section 2 describes the NL-based software tool NL2OCL*via*SBVR; section 3 discuses a case study; section 4 presents evaluation followed by the related work section. The paper ends with a conclusion section.

2. THE NL TO OCL TOOL

The NL2OCL*via*SBVR is a modular NL-based software tool that generates OCL constraints with respect to a target UML model. It takes two inputs: a single English statement and a UML model. To process the input English text first it is linguistically analyzed. In linguistic analysis of the English text, the English text is Parts-Of-Speech (POS) tagged. Then a rule-based parser is used to further process the POS tagged information to extract basic SBVR elements e.g. noun concept, fact type, etc. Here, the SBVR vocabulary is mapped to a SBVR rule. Finally, to generate an OCL expression, the SBVR vocabulary is mapped to OCL syntax using the model transformation approach. The working of these steps has been stated in detail in the following section:

2.1 The Input Documents

NL2OCL*via*SBVR takes two input documents: an English text document and a UML model document. The English text is taken as a plain text file containing only English constraint. Current version of the NL2OCLviaSBVR handles only one English constraint at a time. The given English text should be grammatically correct. UML model is taken as XMI 1.0 format. We used Enterprise Architect to create a UML model and export it in XMI 1.0 format.

2.2 NL to SBVR Transformation

The core of NL2OCLviaSBVR is a NLP module that consists of a number of processing units organized in a pipelined architecture. This NLP module is highly robust and is able to process complex English statements. The NLP system is used to lexically and syntactically process the English text and then perform semantic analysis to identify basic SBVR elements. The core system processes a text into three main processing stages:

2.2.1 Preprocessing

In the preprocessing phase, the input text containing the natural language specification of an OCL constraint for a UML class model is preprocessed for deep processing. Major steps involved in preprocessing phase are splitting the sentences, tokenization, and lemmatization. The preprocessing sub-phases are discussed below:

Sentence Splitting: In first step, the input English text is read and broken into sentences. During sentence splitting, the margins of a sentence are identified and each sentence is separately stored. Sentence splitting is performed using the Stanford parser.

Tokenization: After sentence splitting, each sentence is processed to identify tokens. Again Stanford parser is employed for efficient tokenization. An example is shown in Figure 1:

English:	A customer can place one order.
Tokens:	[A] [customer] [can] [place] [one [order] [.]

Figure 1. Tokenized text using Stanford Parser

Lemmatization: Here, the morphological analysis of words is performed to remove the inflectional endings and to return the base or dictionary form of a word, which is known as the lemma. We identify lemma (base form) in the POS tagged tokens by removing various suffixes attached to the nouns and verbs.

2.2.2 Syntactic Analysis

The output of a typical syntax analysis phase is a tree diagram or other textual representation. Our syntactic analyzer parses the preprocessed text by POS-tagging information and defining the syntactic units also called chunks. In syntax analysis phase, four steps are performed as following:

Parts-of-Speech (POS) Tagging: In this step, parts of speech are identified for each token in the input text. In POS tagging, each token is classified to its respective parts-of-speech category by assigning a specific tag to teach token such as NN, VB, RB, MD, DT, etc. The Stanford POS tagger version 3.0.3 has been used to identify 44 various parts of speech. An example of a POS tagged sentence is shown in Figure 2:

English:	A customer can place one order.
Tokens:	[A/DT] [customer/NN] [can/MD] [place/VB] [one/CD [order/NN] [./.]

Figure 2. Parts-of-Speech tagged text

Generating Syntax Tree: We have used Stanford Parser to generate parse tree. The Stanford parser is a lexically driven probabilistic parser based on Probabilistic Context-Free Grammars (PCFG).

English:	A customer can place one order.
Tokens:	(ROOT (S (NP (DT A) (NN customer)) (VP (MD can) (VP (VB place) (NP (CD one) (NN order)))) (. .)))

Figure 3. Parse Representation

2.2.3 *Semantic Analysis*

A typical semantic analysis yields in a logical form of a sentence. Logical form is used to capture semantic meaning and depict this meaning independent of a particular context. The goal of semantic analysis is to understand the exact meanings of the input text and identify that relationship in various chunks.

Shallow Semantic Parsing: In shallow semantic parsing, the semantic or thematic roles are typically assigned to easy syntactic structure in a NL sentence. This process is also called *Semantic Role Labeling* (SRL). Semantic labeling on a substring (semantic predicate or a semantic argument) in a constraint (NL sentence) 'S' can be applied. Every substring 's' can be represented by a set of words indices as following:

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

Formally, the process of semantic role labeling is mapping from a set of substrings from c to the label set 'L'. Where L is a set of all argument semantic labels,

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

The semantic roles can act as an intermediate representation in NL to SBVR translation. In the context of the targeted representation (SBVR rule representation), we have incorporated the following semantic roles. These semantic roles are typically used in semantic role labeling.

a. Object Type → Common nouns
b. Individual Concept → Proper nouns
c. Verb Concepts → Main Verb
d. Characteristics → Generative Phrases

A sequence of steps was performed for labeling semantic roles to respective semantic predicates. Following are the three main steps involved in the phase of semantic role labeling:

Extracting Semantic Predicates: In this phase, we extract the possible semantic predicates. This module relies mainly on the external resources, thus the elements in target UML Class models (class names, attributes, methods) are likely to be semantic predicates. The chunks not matching the elements of target UML Class model are not semantic predicates or semantic arguments. For extracting semantic predicates we check if the verb is a simple verb, a phrasal verb or a verbal collocation and locate the verb in (see Figure 4).

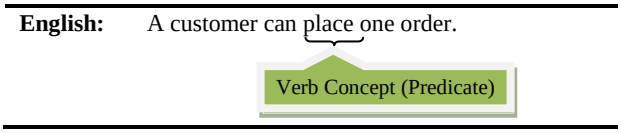

Figure 4. Identifying Verb concepts (Predicate)

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

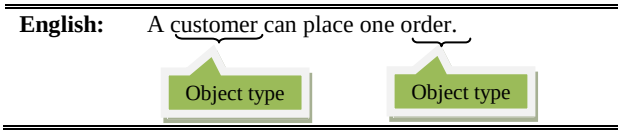

Figure 5. Identifying semantic arguments

Extracting Semantic Arguments: In English sentences, object type can be represented with pre-modifiers such as articles (determiners) and with post-modifiers: prepositional phrases, relative (finite and non-finite) clauses, and adjective phrases.

Semantic Interpretation: In lexical semantics, the frame is also considered a useful tool in text semantics and the semantics of grammar. The interpreter of a text invokes a frame when assigning an interpretation to a piece of text by placing its contents in a pattern known independently of the text. A text evokes a frame when a linguistic form or pattern is conventionally associated with that particular frame. Figure 7 shows an example of the semantic interpretation we have used in the presented approach for NL to OCL transformation.

English: A customer can place one order.

Logical: (place
(object_type = (the ~ (customer ? x))
(object_type = (the ~ (order ? y)))

Figure 6. Semantic roles assigned to input English sentence.

The output of the NLP module is an xml file that contains the parsed English text with all the extracted information.

Deep Semantic Parsing: In natural languages, quantifications are typically expressed with noun phrases (NPs). However, in First-Order Logic (FOL), the variables are quantified at the start of the logical expressions. Generally, the natural language quantifiers are much more vague and varied. This vagueness makes translation of NL to FOL complex. However, we have set of heuristic rules to identify the quantifications:

i. Universal Quantification (∀X): The universal quantification is mapped to *Universal Quantification* in SBVR. The NL quantification structures 'each', 'all', and 'every' are mapped to universal quantificational structures. Similarly, the determiners 'a' and 'an' used with the subject part of the sentence are treated as universal quantification (see Figure 7).

ii. Existential Quantification (∃X): The existential quantification is mapped to *Existential Quantification* in SBVR. The keywords like many, little, bit, a bit, few, a few, several, lot, many, much, more, some, etc. are mapped to existential quantification.

ii. Uniqueness Quantification ($\exists_{=1}$X)*:* The uniqueness quantification is mapped to *Exactly-One* Quantification in SBVR. The determiners 'a' and 'an' used with object part of the sentence are treated as uniqueness quantification.

iii. Solution Quantification (§X)*:* The solution quantification is mapped to *Exactly-n* Quantification in SBVR. If the keywords like more than or greater than are used with *n* then solution quantifier is mapped to *At-most Quantification* (see Figure 7). Here, if the terms "less than" or "smaller than" are used with *n* then solution quantifier is mapped to *At-least Quantification*.

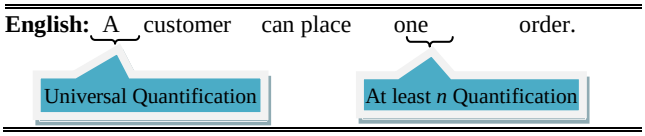

Figure 7. Identifying quantifications

The SBVR produces a SBVR rule in the form of text string that is further formatted using the SBVR notation i.e. Structured English

described in the section 2.3.4. The output SBVR module is saved and exported in two separate files: an xml file contains the SBVR vocabulary and respective details; a text file contains the formatted SBVR rule.

2.3 The SBVR to OCL Transformation

The OCL module maps a SBVR rule to an OCL expression by using model transformation that incorporates the mapping rules between SBVR and OCL metamodels. SBVR to OCL mapping rules typically define the conversion of element(s) of the SBVR metamodel to equivalent element(s) of the OCL metamodel. In OCL module, SiTra [17] library is used to implement model transformation. OCL module uses the output of the SBVR module i.e. the SBVR vocabulary to generate an OCL expression. A set of mapping rules were defined to map the SBVR vocabulary to different type of OCL expressions e.g. invariant, precondition and post condition. OCL queries have not been supported by the current version of the NL2OCLviaSBVR. A brief description of the mapping rules is provided in the following section:

2.3.1 OCL Package and Context

For any type of OCL expression, two elements are basic requirements: package and context. The UML package is mapped to the OCL package. While, the context of an OCL expression defines the scope of the given invariant or pre/post condition. To specify the context of an OCL invariant, the major actor in the SBVR rule is extracted to specify the context. To specify the context of an OCL pre/post condition, the action performed by the actor in a SBVR rule is considered as the context.

2.3.2 Mapping OCL Constraints

Transformation rules for mapping of UML-SBVR specification to OCL constraints are defined in this section. There are two basic types of an OCL constraints; invariant of a class, and pre/post condition of an operation. Constraint on a class is a restriction or limitation on a particular attribute, operation or association of that class with any other class in a model [18].

2.3.3 Mapping OCL Invariants

The OCL invariant specifies a condition on a class's attribute or association. Typically, an invariant is a predicate that should be TRUE in all possible worlds in UML class model's domain. The OCL context is specified in the invariants by using self keyword in place of the local variables.

2.3.4 Mapping OCL Pre/Post Conditions

Similar to the OCL invariant, the OCL preconditions and the OCL postcondition are used specify conditions on operations of a class. Typically, a precondition is a predicate that should be TRUE before an operation starts its execution, while a postcondition is a predicate that should be TRUE after an operation completes its execution [16].

2.3.5 Mapping OCL Expressions

The OCL expressions express basic operations that can be performed on available attributes of a class. An OCL expression in the OCL invariant can be used to represent arithmetic, and logical operations. OCL arithmetic expressions are based on arithmetic operators e.g. '+', '–', '/', etc., while, logical expressions use relational operators e.g. '<', '>', '=', '<>', etc. and logical operators e.g. 'AND', 'implies', etc.

2.3.6 Mapping OCL Operations

The OCL collections represent a set of attributes of a class. A number of operations can be performed on the OCL collections e.g. sum, size, forAll(), count, isEmpty, etc.

All the defined transformation rules were implemented in a java based library SiTra (*Si*mple *Tra*nsformation). The output of the OCL module is a complete OCL expression. The output OCL is saved and exported in a separate text file.

3. A CASE STUDY

Here a case study is discussed from the domain of UML modeling. The case study was originally presented by Alanna Zito in her MSc thesis [25] with Juergen Dingel to create an encoding of PackageMerge constraints in Alloy. We want to formalize PackageMerge constraints in OCL using our tool NL2OCLviaSBVR. The problem statement of the case study is based on the constraints of package merge rules given in clause 7.3.40 in UML 2.3's Superstructure specification document. The problem statement is based on constraints and transformations for association rules that are as below [8]:

> *1. The rules only apply to binary associations.*
> *2. The receiving association end must be a composite if the matching merged association end is a composite.*
> *3. The receiving association end must be owned by the association if the matching merged association end is owned by the association.*

The transformations defined for association rules are [8]:

> *1. A merge of matching associations is accomplished by merging the Association classifiers and merging their corresponding owned end properties according to the rules for properties and association ends.*
> *2. For matching association ends: if neither association end is navigable, then the resulting association end is also not navigable. In all other cases, the resulting association end is navigable.*

Section 4 presents the results produced by the NL2OCLviaSBVR.

3.1 Implementation Details

The problem statement of the used case study was processed by using our tool NL2OCL*via*SBVR. Following are the results of one precondition and postcondition defined in the problem statement. For package merge preconditions (i.e. package merge constraints for the association rules) and the postconditions (i.e. package merge transformations for the association rules), the screen shots of output windows for SBVR rules and OCL constraints have been shown in Figure 8, 9, 10, and 11:

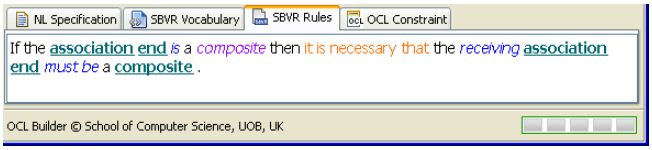

Figure 8. English to SBVR mapping of the costraints of association rules

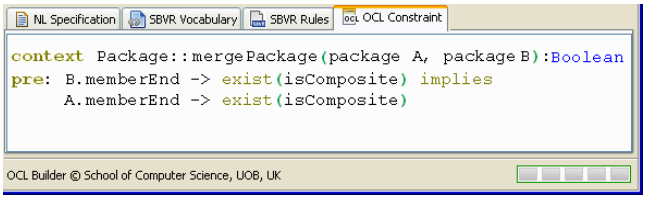

Figure 9. SBVR to OCL mapping of the costraints of association rules

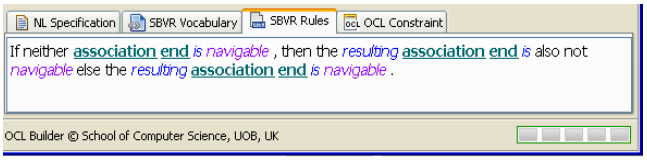

Figure 10. English to SBVR mapping of the transforamtions of association rules

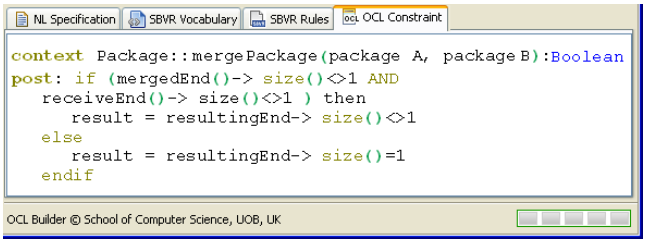

Figure 11. SBVR to OCL mapping of the transformations of association rules

The above shown screenshots are from our tool NL2OCLviaSBVR that gets a UML Model and English representation of a package merge constraint or transformation.

4. EVALUATION

To test the accuracy of the OCL constraints generated by the designed system two classes were defined: preconditions and post-conditions. The package merge English constraints and transformation were classified into three classes with respect to complexity levels of input i.e. simple, compound and complex.

To test tools accuracy 10 examples of each complexity-level were used. Constraint types for each 10 examples were generated. Each generated OCL constraint from each category was type-checked. For type checking OCLarity tool was used that is an OCL type checking tool. For the sake of type checking in OCLarity, the used class model and the generated OCL constraint were given as input. A matrix of results of generated diagrams is shown below.

Table 1: Evaluatin results

Complexity level/ Constraint Type	Precondition	Post condition	Total
Simple	91.5%	89.3%	91.63%
Complex	90.2%	87.8%	89.73%
Compound	84.7%	79.8%	83.7%

Average accuracy: 88.33%

A matrix representing OCL constraints accuracy test (%) for pre and post conditions is constructed. Overall accuracy for all types of OCL constraints is determined by adding total accuracy of all categories and calculating its average that is 88.33%.

4.1 Usability Survey

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

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

Each group consists of 10 users. A set of inputs such as English specification of OCL constraints were provided to all the users. First all the users were asked to solve the input manually and then they were asked to generate the OCL constraints by using our tool NL2OCLviaSBVR. Once all the users finished their work they were given a questionnaire to fill. In the questionnaire, questions were asked regarding various aspects: simple to use, time-saving, correctness, etc. Each user was asked to give 1 to 10 score for each category.

Table 2. Usability Survey Results

User	***Easy to Use***		***Time-Saving***		***Correctness***	
	Manual	By Tool	Manual	By Tool	Manual	By Tool
Novel	30%	90%	25%	85%	15%	65%
Medium	55%	85%	40%	80%	50%	70%
Expert	70%	85%	60%	70%	80%	80%
Average	**51.66%**	**86.66%**	**41.66%**	**78.33%**	**48.33**	**71.66%**

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

5. CONCLUSION & FUTURE WORK

This research paper presents a framework for dynamic generation of the OCL constraints from the NL specification provided by the user. Here, the user is supposed to write simple and grammatically correct English. The designed system can find out the noun concepts, individual concepts, verbs and adjectives from the NL text and generate a structural or behavioral rule according to the nature of the input text. This extracted information is further incorporated to constitute a complete SBVR rule. The SBVR rules are finally translated to OCL expressions. SBVR to OCL translation involves the extraction of OCL syntax related information i.e. OCL context, OCL invariant, OCL collection, OCL types, etc. and then the extracted information is composed to generate a complete OCL constraint, or pre/post-condition.

As this paper aims to address a major challenge related to usability of OCL, we have presented a method of applying model transformations to create OCL statement from Natural Language expressions. The presented transformation makes use of SBVR as an intermediate step to highlight the syntactic elements of natural languages and make NL controlled and domain Specific. The use of automated model transformations ensures seamless creation of

OCL statements and deemed to be non-intrusive. As a next step, we are hoping to investigate usability aspects of the tool directly via empirical methods involving teams of developers.

6. REFERENCES

[1] OMG. 2007. Unified Modeling Language (UML), *OMG Standard*, v. 2.3.

[2] OMG. 2006. Object Constraint Language (OCL), *OMG Standard*, v. 2.0.

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

[4] Wahler M. 2008. Patterns to Develop Consistent Design Constraints, *PhD Thesis*, ETH Zurich, Switzerland.

[5] Cabot, J. 2006. Ambiguity issues in OCL postconditions. In: Proc. *OCL for (Meta-) Models in Multiple Application Domain - MODELS'06*, Technical Report.

[6] Correa A., Werner C., Barros M. 2007. An Empirical Study of the Impact of OCL Smells and Refactorings on the Understandability of OCL Specifications, *MODELS'07, LNCS 4735*. pp 76-90

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

[8] OMG, 2007. UML Superstructure specification document, *OMG Standard*, v. 2.3.

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

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

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

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

[13] Linehan M. 2008. SBVR Use Cases. *Int. Symposium on Rule Representation, Interchange and Reasoning on the web*, *RuleML, LNCS* Vol.5321 pp. 182-196

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

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

[16] Cabot J., et al. 2009. UML/OCL to SBVR Specification: A challenging Transformation, *Journal of Information systems*

[17] Akehurst, D.H., Boardbar, B., Evans, M., Howells, W.G.J., McDonald-Maier, K.D. 2006. SiTra: Simple Transformations in Java , *ACM/IEEE 9TH International Conference on Model Driven Engineering Languages and Systems,LNCS,* Vol. 4199, pages 351-364

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

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

[20] Bajwa, I. S., Choudhary M.A. 2006. A Rule Based Paradigm for Speech Language Context Understanding. *International Journal of Donghua University* (English Edition). 23, 06 (June 2006), 39-42.

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

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

[23] Demuth B, Wilke C. 2009. Model and Object Verification by Using Dresden OCL. I*n R.G. Workshop on Innovation Information Technologies: Theory and Practice*, pp. 81-89

[24] IBM OCL Parser, Sep 2009 http://www-01.ibm.com/ software/awdtools/library/standards/ocl-download.htm

[25] Jürgen D., Zinovy D., Alanna Z. 2008. Understanding and improving UML package merge. *SoSyM*, 7(4):443-467

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

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

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

[29] Kovacs L., Kovasznai G., Kusper G. 2008. Metamodels in Generation of UML Using NLI-Based Dialogue. In *5th International Symposium on ASCII*, pp. 29-33

[30] Bryant B., et al. 2008. From Natural Language Requirements to Executable Models of Software Components. In *Workshop on S. E. for Embedded Systems* pp.51