

Paraphrasing OCL Expressions with SBVR

Raquel Pau¹ and Jordi Cabot²

¹ GTD Ingeniería de Sistemas y de Software
raquel.pau@gtd.es

² Estudis d'Informàtica, Multimedia i Telecomunicació, Universitat Oberta de Catalunya
jcabot@uoc.edu

Abstract. A conceptual schema (CS) should be explained to the stakeholders to validate that it is an appropriate representation of all knowledge of the domain. One of the best ways to explain the CS is to describe it by means of natural language expressions (*paraphrasing*). Even though paraphrasing has been studied for the most typical elements of a CS, current methods are, in general, unable to cope with the textual business rules that complement the CS. In this paper, we cover this gap by presenting a method that generates natural language explanations for business rules expressed in OCL (Object Constraint Language), the standard language to specify business rules on UML-based CSs. As an intermediate step, our method translates the OCL expression into a SBVR (Semantics of Business Vocabulary and Business Rules) representation.

1 Introduction

The specification of an information system must include a formal representation of the knowledge of the domain required by the system to perform its functions. In conceptual modelling, this representation is known as its conceptual schema (CS). A CS must include the definition of all relevant business rules of the domain.

To validate the correctness of the CS we must rely on the external stakeholders, as the experts in the domain. Nevertheless, to do so, customers must be able to understand the knowledge represented in the CS. To facilitate this task, designers should reexpress the CS in a way that they can easily understand. Typically, one of the most useful alternatives is to describe the elements of the CS by means of natural language expressions. This process is known as *paraphrasing*.

The OMG has recently proposed the SBVR (Semantics of Business Vocabulary and Business Rules [4]) specification as a way to facilitate the translation among a set of different languages. SBVR is specially suited for acting as an intermediate step in a CS to natural language transformation since it is conceptualized optimally for business people.

In this sense, the goal of this paper is to introduce an automatic translation from UML-based CSs to SBVR, where the initial UML schemas [5] are complemented with textual expressions in OCL (Object Constraint Language [3]) to define all business rules that cannot be graphically represented. In particular, since the own SBVR standard already sketches the translation of the graphical UML elements that may appear in the CS (classes, associations, attributes and so forth), in this paper we

will focus on the translation of its OCL business rules. Then, the obtained SBVR representation can be presented to the customers as a list of self-explaining natural language expressions using the predefined alternative notations to express SBVR concepts by means of English statements (either in the SBVR Structured English style or using the RuleSpeak approach [4]) included in the SBVR standard.

As far as we know, ours is the first proposal to provide such translation. To the best of our knowledge, only [1] is aimed at generating natural language sentences from OCL expressions. However, this approach provides an ad-hoc solution instead of relying in a standard format for defining the business rules, as SBVR.

The rest of the paper is structured as follows. Section 2 presents our preliminary OCL to SBVR translation. In Section 3, we generate natural language expressions from the resulting SBVR excerpts. Finally, Section 4 presents some conclusions.

2 From OCL to SBVR

OCL is a formal high-level language used to specify textual business rules that cannot be expressed using the graphical constructs provided by the UML.

In OCL, invariants (i.e. static constraints) are defined in the context of a specific type, called the *context type* of the constraint. The actual OCL expression stating the constraint condition is called the *body* of the constraint. The body is a boolean expression that must be satisfied by all instances of its context type. As an example, in constraint *ownersDrive* (Fig. 1) *Car* is the context type, the variable *self* refers to an instance of *Car* and the *exists* condition (the body) must hold for all possible values of *self* (i.e. for all instances of *Car*).

The aim of this section is to introduce a method for the translation of OCL constraints¹ into SBVR logical formulations. A logical SBVR formulation structures the meaning of a rule using several propositions. Different kinds of simple propositions are predefined, as quantifications, logical operations (conjunction, disjunction, equivalence...) and projections, which can be roughly defined as logical formulation of a condition that results in a set or a bag (multi set) of instances. Fig. 1 shows the logical formulation for the previous *ownersDrive* constraint.

As a first step, each constraint is expressed as a new SBVR logical formulation. In particular, they are translated as *necessities*. Given a constraint *c* with a body *b* and defined over a context type *t*, the equivalent SBVR representation for *c* would be:

The rule is a proposition meant by a necessity claim.
. The necessity claim embeds a universal quantification. (to ensure that all instances of t satisfy b)
.. The universal quantification introduces a variable. (the self variable)
... The variable ranges over the concept 't'. (self will be mapped to each different t instance)
.. The universal quantification scopes over ...'b condition'....

Then, the body *b* of the OCL constraint is translated into an appropriate SBVR representation using a set of translation patterns, where each pattern defines how to translate a different construct² that may appear in an OCL expression. The full OCL

¹Note that not all kinds of OCL expressions can be directly mapped to SBVR concepts. Required extensions to SBVR to achieve a full translation are left as further work.

²To simplify the process, complex OCL constructs are first rewritten in terms of basic ones [2].

expression translation is performed by combining a set of appropriate patterns depending on the expression structure. We assume that the UML schema has already been translated to SBVR (e.g. following the guidelines provided in the own SBVR standard) before translating the OCL expressions themselves.

Translation of Basic OCL operators

The following patterns describe the translation of basic OCL operators. In the patterns, X and Y represent variables or constant literals of the appropriate type. The sentence “*previous translation*” refers to the result of translating a previous part of the OCL expression. “*Translation of X*” and “*translation of Y*” refer to the result of translating the OCL expressions corresponding to the X and Y variables. For each pattern we define the target OCL expression and its SBVR translation. The dots before each proposition indicate the proposition nesting level in complex expressions.

- *X implies Y*
The previous translation introduces an implication.
.The implication has an antecedent that is a... translation of X ...
.The implication has a consequent that is a ... translation of Y ...
- *X and/or/xor Y*
The previous translation introduces a [conjunction \ disjunction \ exclusive disjunction].
. The [conj./disj./excl. disj.] has a logical operand₁ that is a... translation of X ...
. The [conj./disj./excl. disj.] has a logical operand₂ that is a... translation of Y ...
- *Not X*
The previous translation introduces a logical negation.
. The logical negation has a negand that is a... translation of X ...
- *X [< | > | =] Y*
The previous translation introduces an atomic formulation
. The atomic formulation is based on the fact type quantity₁ [is less than \ is greater than\ is] quantity₂
. The atomic formulation has the first role binding.
.. The first role binding is of the role quantity₁
.. The first role binding binds to the ... translation of X ...
. The atomic formulation has the second role binding.
.. The second role binding is of the role quantity₂
.. The second role binding binds to the... translation of Y ...
- *X.oclIsTypeOf (T)*
The previous instantiation introduces an instantiation formulation.
. The instantiation formulation considers the concept ‘T’.
. The instantiation formulation binds to the... translation of X ...

Translation of navigation expressions

In OCL, the *dot* notation applied over an object (or collection of objects) allows to navigate from that object to related objects in the system using the associations defined in the CS. That is, given an object o of type t_1 and a type t_2 , related with t_1 through an association *assoc* with roles r_1 and r_2 , respectively, the expression $o.r_2$ returns the set of objects of t_2 related with o . This expression is translated as follows:

The first universal quantification introduces the first variable (the initial object o)
. The first variable scopes over the concept ' t_1 '
The first universal quantification scopes over a second universal quantification
. The second universal quantification introduces the second variable
.. The second variable scopes over the concept ' t_2 '
. The second universal quantification is restricted by an atomic formulation
.. The atomic formulation is based on the fact type 'assoc'
... The atomic formulation has the first role binding.
... The first role binding is of the role ' r_1 '
... The first role binding binds to the first variable
.. The atomic formulation has the second role binding.
... The second role binding is of the role ' r_2 '
... The second role binding binds to the second variable
. The second universal quantification scopes over ... translating the rest of the expression.

Note that, in SBVR, navigations are expressed by defining a second variable of type t_2 (to represent the set of objects retrieved by the navigation) and stating that this second variable returns the subset of instances of t_2 related with the first variable (of type t_1). Access to attributes or association classes is translated in the same way.

Translation of collection expressions

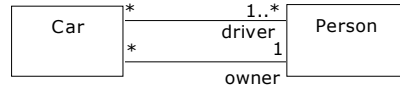
In OCL, a collection C of objects of type T can be manipulated through a predefined set of OCL operations and iterators. Each expression introduces a new quantification (that in some cases may be merged with the translation of the previous part of the OCL expression). Their basic translation is the following:

- $C \rightarrow \text{exists}(v \mid \text{boolean-condition-over-}v)$
The previous translation of C introduces an at-least-one quantification
. The at-least-one quantification introduces a variable. (v)
.. The variable ranges over the concept T
. The at-least-one quantification scopes over the translation of boolean-condition-over- v
- $C \rightarrow \text{forAll}(v \mid \text{boolean-condition-over-}v)$
The previous translation of C introduces a universal quantification
. The universal quantification introduces a variable. (v)
.. The variable ranges over the concept T .
. The universal quantification scopes over the translation of boolean-condition-over- v
- $C \rightarrow \text{select}(v \mid \text{boolean-condition-over-}v)$ ³
The previous translation of C introduces a universal quantification
. The universal quantification introduces a variable
.. The variable ranges over the concept T
. The universal quantification scopes over the translation of boolean-condition-over- v
- $C \rightarrow \text{iterate}(v [\text{acc}:T'] \mid \text{expression-with-}v)$
The previous translation of C introduces a universal quantification
. The universal quantification introduces a variable (v)
.. The variable ranges over the concept T
. The universal quantification scopes over a second universal quantification
.. The second universal quantification introduces a second variable (acc)
... The second variable ranges over the concept T'
.. The second universal quantification scopes over the translation of expression-with- v

³ The translation of a *select* expression will differ depending on the context where it is applied. The one provided herein is useful when the *select* is the last OCL operator in the expression.

Original OCL expression:

```
context Car inv ownersDrive:
self.driver->exists(d|d=self.owner)
```

**1) Semantic formulation in SBVR:**

The rule is a proposition meant by a necessity claim. (*ownersDrive*)

. The necessity claim embeds a universal quantification.

.. The universal quantification introduces a first variable. (*self*)

... The first variable ranges over the concept 'car'.

.. The first universal quantification scopes over an at-least-one quantification. (*exists*)

... The at-least-one quantification introduces a second variable. (*d, i.e.: self.driver*)

.... The second variable ranges over the concept 'person'.

.... The at-least-one quantification is restricted by an atomic formulation.

..... The atomic formulation is based on the fact type 'driver of car'.

..... The atomic formulation has the first role binding.

..... The first role binding is of the role 'car'.

..... The first role binding binds to the first variable.

..... The atomic formulation has the second role binding.

..... The second role binding is of the role 'driver'.

..... The second role binding binds to the second variable.

.... The at-least-one quantification scopes over a third universal quantification.

..... The third universal quantification introduces a third variable. (*self.owner*)

..... The third variable ranges over the concept 'person'

..... The atomic formulation is restricted by the fact type 'owner of car'.

..... The atomic formulation has the first role binding.

..... The first role binding is of the role 'car'.

..... The first role binding binds to the first variable.

..... The atomic formulation has the second role binding.

..... The second role binding is of the role 'owner'.

..... The second role binding binds to the third variable

.... The third universal quantification scopes over an atomic formulation.

..... The atomic formulation is based on the fact type thing₁ = thing₂

..... The atomic formulation has the first role binding.

..... The first role binding is of the role 'thing₁'.

..... The first role binding binds to the second variable.

..... The atomic formulation has the second role binding.

..... The second role binding is of the role 'thing₂'.

..... The second role binding binds to the third variable

2) Structured English expression:

It is necessary that at least one driver of a car is the owner of the car

Fig. 1. Example of an: 1) OCL to SBVR and 2) SBVR to natural language translation. Next to each SBVR proposition, we show the subset of the OCL expression it refers to.

3 From SBVR to Natural Language

Once we have the SBVR-based representation of the OCL rules, the predefined mappings provided in the SBVR standard can be used to generate natural language descriptions of the rules in the *Structured English* or *RuleSpeak English* notations.

Roughly, to automate this part of our translation process we proceed as follows. First, the initial ‘*necessity claim*’ proposition generates an ‘*It is necessary that*’ sentence. Second, we search for the most inner quantification q in the SBVR representation. Let lf be the logical formulation that q scopes over. Then, the translation continues by translating lf as well as by (recursively) translating all variables involved in lf and q according to the appropriate patterns described in [4]. See Fig. 1 for an example.

4 Conclusions and Further Work

We have presented a method to facilitate the validation of business rules specified in the UML/OCL languages. Our method uses the SBVR standard to bridge the gap from UML/OCL to natural language, the only representation that most customers are able to understand, and thus, validate. We believe our paper can also be a first step towards a tighter integration of the business rules and UML/OCL communities.

As a further work we would like to propose a set of extensions to the SBVR standard to be able to translate all kinds of OCL expressions. We would also like to test our approach in industrial case studies and use the customers’ feedback to refine and improve the quality of the final natural language expressions.

Acknowledgements

Thanks to Ruth Raventós for their useful comments to previous drafts of this paper. This work was partially supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIN2005-06053.

References

1. Burke, D.A., Johannisson, K.: Translating Formal Software Specifications to Natural Language: A Grammar-Based Approach. In: Blache, P., Stabler, E.P., Busquets, J.V., Moot, R. (eds.) LACL 2005. LNCS (LNAI), vol. 3492, pp. 51–66. Springer, Heidelberg (2005)
2. Cabot, J., Teniente, E.: Transformation Techniques for OCL Constraints. *Science of Computer Programming* 68(3), 152–168 (2007)
3. OMG: UML 2.0 OCL Specification. OMG Adopted Specification (ptc/03-10-14) (2003)
4. OMG: Semantics of Business Vocabulary and Rules (SBVR) Specification. OMG Adopted Specification (drc/06-03-01) (2006)
5. OMG: UML 2.1.1 Superstructure Specification. OMG Adopted Specification (formal/07-02-03) (2007)