Eliminating Qualifier and Association Class Ambiguities from OCL*

Stephan Flake
C-LAB, University of Paderborn
Fuerstenallee 11, 33102 Paderborn, Germany

1 Introduction

This paper presents different proposals to eliminate ambiguities from OCL expressions with respect to issue #3513, published in the current documentation of recent OCL changes. It has been recognized that qualifiers and association classes cannot be distinguished in certain OCL expressions. This is due to the fact that it is allowed to use the same name more than once in the scope of different classifiers in a single class diagram. Although this problem does not seem to appear frequently and can be regarded to be of minor severity, it is still necessary to resolve this conflict, as OCL claims to be a precise, unambiguous language (cf. [4], page 8). Without this, OCL expressions cannot be correctly parsed and validated by CASE tools. On the other hand it has to be noted that OCL should still be understood by practitioners, which limits the extent of introducing new notations and/or semantics.

After describing the general problem in the next section, an example is presented in section 3 by a class diagram and a sample OCL constraint. Some proposals to resolve the problem are presented in section 4. The paper is completed by a brief conclusion in section 5.

2 The Issue

Issue #3513 is described in the document of recent OCL changes [2] as follows:

> ***Descriptor:*** *OCL: qualifiers and association classes ambiguity*
> ***Source:*** *unknown*
> ***Reference:*** *Section 7- OCL*
> ***Nature:*** *Revision*
> ***Severity:*** *Minor*
> ***Summary:*** *Qualifiers, written in brackets after the path name of a feature call, can express two different things. - qualifying use: A qualifier is used to give the qualifying value of a qualified association (chapter 7.5.7). - navigational use: A qualifier is used to refine the navigation to association classes. While this navigational use is necessary only with recursive associations, it is legal for every navigation to an association class (chapter 7.5.5). There is no way to distinguish these two sorts of qualifiers. There are even expressions where both uses of the qualifiers would be necessary at once, but this problem is restricted to such models that contain a recursive, qualified association that has an association class. [...]*

* This work is supported by a German research grant (DFG, SPP 1064: Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen)

3 Example

We do not follow the example originally presented with this issue, as it does not consider a class diagram in which navigational use is truly required. Instead, a different example should demonstrate the ambiguity with respect to a recursive qualified association. In figure 1, two classes Bank and Person are connected by an association with rolenames employer and employees. Connected to class Person there is a recursive association with rolenames bosses and subordinates, qualified by an integer attribute score in the association class EmployeeRanking (cf. [3], paragraph 7.5.5).

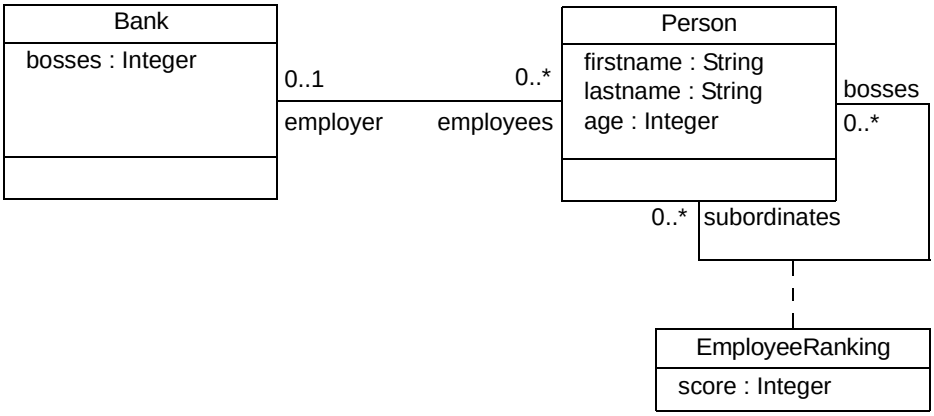

Figure 1. UML class diagram

The name of the association class alone is not sufficient for navigation in recursive associations. Additionally, the direction in which the association is navigated has to be specified. Let us consider a navigation to the association class EmployeeRanking towards the bosses end. According to paragraph 7.5.5 in [3], the rolename of the direction is added to the association class name inside square brackets. In the constraint

```
context Person inv:
employeeRanking[bosses]->size > 0
```
(1)

the sub-expression employeeRanking[bosses] evaluates to the set of EmployeeRankings for the collection of bosses. Note, that the unqualified use of the association class name is not permitted in such a recursive association. Now consider the constraint

```
context Bank inv:
employees.employeeRanking[bosses]->size > 0
```
(2)

in the context of class Bank, which seems to be very similar to constraint (1). Here, the use of bosses can either denote the integer attribute of class Bank or the association end of the recursive association of class Person. In the first case, employees.employeeRanking[bosses] evaluates to the set of EmployeeRankings whose integer value is equal to the integer value bosses of class Bank, while in the other case the result is - as in (1) - the set of EmployeeRankings belonging to the collection of bosses. In general, these sets are not equal.

4 Possible Solutions

Taking a closer look at the UML class model and the OCL grammar, we find the following causes for the illustrated ambiguity:

1. UML allows to use the same name in different model elements of a single class diagram under certain conditions. For instance, in the context of the considered issue an attribute in one class can have the same name as a qualified association end in the scope of another class.
2. In the OCL grammar, there is one production rule for qualifiers, leading to an actual parameter list embraced by square brackets. Actual parameters can syntactically be all kind of OCL logical expressions, in particular attributes and qualified association ends.
3. In the OCL chapter of the UML specification document [3], no semantics are given which restrict the formulation of ambiguous OCL expressions in the considered issue.

It seems to be not possible to eliminate the first listed cause without restricting the UML metamodel of class diagrams. But instead of proposing a new semantics for class diagrams, we should rather concentrate on the other two causes in order to find a solution within OCL.

In the remainder of this section we investigate whether this problem can be resolved on the level of the OCL grammar, or by additional rules of use, or whether it should be left to the modelers. In each of these approaches we outline different possible solutions and briefly discuss their suitability and drawbacks.

4.1 Modification of Grammar Elements

Some grammar terminals could be replaced or introduced to explicitly distinguish between different semantics. For our problem, the square brackets are obvious candidates. The relevant production rules to analyze are taken from the current Draft 1.4 OCL Grammar [1]:

```
propertyCall          :=  pathName timeExpression?
                          qualifiers? propertyCallParameters?
qualifiers            :=  "[" actualParameterList "]"                    (3)
actualParameterList   :=  expression ( "," expression )*
```

One way to resolve the problem is by replacing the qualifiers production rule. First, a split into two parameter lists is introduced:

```
qualifiedActualParameterList   :=  expression ( "," expression )*
navigatedActualParameterList   :=  expression ( "," expression )*      (4)
```

These two lists can then be used in the qualifiers production rule in the following ways.

(a) Keep the embracing brackets and introduce a separator:

```
qualifiers  :=  "[" navigatedActualParameterList "|" qualifiedActualParameterList "]"
                | "[" navigatedActualParameterList "]"                 (5)
                | "[" "|" qualifiedActualParameterList "]"
```

(b) Introduce an additional construct:

```
qualifiers   :=   ( "<" navigatedActualParameterList ">" )?
                  ( "[" qualifiedActualParameterList "]" )?
```
(6)

(c) A shorter alternative to (b) that produces the same language:

```
propertyCall          :=   pathName timeExpression?
                           ( "<" navigatedActualParameterList ">" )?
                           ( "[" qualifiedParameterList "]" )?
                           propertyCallParameters?
```
(7)

Discussion. All proposals distinguish between navigated and qualified parameter lists. This leads to a redundant production rule, which (informally) clarifies the different meanings of the two lists. Proposal (5) can lead to *empty sections* if only one parameter list is applied, e.g. [| employees] . Empty sections should be avoided, as they complicate the readability of OCL expressions. The preferable version is (6) - or alternatively (7) - as it does not produce empty sections. This proposal is keeping the brackets for qualified parameters, like it is widely understood by practitioners. Thus, a new notation is used to *visually* separate navigation to an association class from access via a specific qualifying value. With this notation, not only the conflict is solved, but there are even more complex expressions possible. For instance, in the expression

```
context Bank inv:
employees.employeeRanking<bosses>[bosses]->size > 0
```
(8)

the part employees.employeeRanking<bosses> evaluates to the set of employeeRankings of the collection of bosses, and then [bosses] is additionally restricting this set to the employeeRankings whose score is equal to the value of Bank.bosses.

Further production rules for the two types of actual parameter lists could state that they can only contain appropriate elements, namely rolenames or attributes/values. But this regards to the semantics and not to the domain of the grammar.

4.2 Providing an Identifying Context

Generally, a different format of the potentially ambiguous actual parameters can be applied in order to resolve the regarded problem.

(a) A first idea to unambiguously access a diagram element in OCL expressions is to provide its whole path at all times, starting from the context with self. Although this is not necessary in most cases, it is essential for names which are potentially ambiguous, like in figure 1. In OCL expressions with context Bank the name bosses must then either be specified as self.bosses or as self.employees.bosses.

(b) Another idea dedicated to the domain of OCL is to explicitly attach a *model element type* to the regarded parameters. With respect to figure 1 we could either state bosses:Rolename or bosses:Attribute, like in the following expression:

```
context Bank inv:
employees.employeeRanking[bosses:Attribute]->size > 0
```
(9)

Discussion. An advantage of proposal (b) is that the information is very simple to extract from the class diagram. This approach requires changes in the OCL grammar: Actual parameters are specified together with model element types which are elements of the set {Attribute, Operation, Rolename, Classifier}. Note that the ambiguity is only resolved if this sort of extension was required for all parameters. Therefore, this approach might be too complex for practise. Proposal (a) might also be a task too cumbersome; it is even worse than proposal (b), as a complete navigation along several associations results in quite long expressions.

4.3 Precedence Rules

Names in OCL expressions can generally represent different elements of the given class diagram. For instance, they can denote rolenames, classifiers, attributes or operations. As UML allows to use the same name multiple times in the same class diagram, it will consequently lead to misinterpretations. As long as class diagrams are discussed by human modelers, they might additionally point at the elements, thus avoiding an ambiguous interpretation. Discussing a diagram without physically pointing at its elements is much more difficult, as the context of names must then be explicitly stated. In OCL, we also cannot physically point at the diagram elements. Solutions to apply a unique identifying context by syntactically extending OCL have been discussed in the previous sections. Now we investigate an approach with precedence rules that does not require any changes to the OCL syntax. One of the rules can be added to the relevant paragraphs in the official UML specification (paragraphs 7.5.5 and 7.5.7):

> **Precedence Rule 1.** *Given an OCL expression within the context of a classifier C, local names of C have a higher precedence than names of other diagram elements.*

Or, contrary to the first rule:

> **Precedence Rule 2.** *Given an OCL expression, each occurring name belongs to the scope of its preceding expression element.*

Of course, these are just informal rules, and it must be clearly defined what is being understood as *name, local name* and *preceding expression element* in an OCL expression. With rule 1, the original expression (2) evaluates to the set of EmployeeRankings with the value of score equal to the value of Bank.bosses. If the rolename bosses was intended to be specified, the expression must now explicitly state

```
context Bank inv:
employees.employeeRanking[self.employees.bosses]->size > 0
```
(10)

When rule 2 should be applied, the original expression (2) must be changed to

```
context Bank inv:
employees.employeeRanking[self.bosses]->size > 0
```
(11)

Discussion. Introducing precedence rules has the advantage that the syntax does not have to be changed. The former ambiguous notation is getting a unique semantics, and alternatives have to be expressed by providing a complete path. But users have to know the additional precedence rules in order to correctly use OCL. From the user's point of view it would be better if ambiguous notations are syntactically not possible at all.

4.4 Leaving the Issue to the Modelers and Tools

Finally, this paragraph presents a solution which does not directly affect OCL syntax and semantics. As long as no other solution is found, the OCL specification document should inform users about the problem. A first advice should be placed at the end of paragraph 7.5.5 in the official UML specification together with an example. For instance, the advice could start like this:

> **Advice 1.** *Note that there might occur an ambiguity in correlation with qualified attribute values (cf. paragraph 7.5.7). There are class diagrams possible in which a name is both representing an attribute in one class and a rolename in the scope of another class. To avoid ambiguities in OCL expressions, either rename these elements in the diagram or clearly state in the OCL expression which name is meant by applying the complete path.*

There should be a another advice at the end of paragraph 7.5.5, e.g.

> **Advice 2.** *Note that there might occur an ambiguity in correlation with navigation to association classes via rolenames (cf. paragraph 7.5.5). [...]*

Parsers that follow this approach should also be aware of this matter and raise an exception or should ask modelers to select one of the alternatives.

Discussion. This approach can only be an interim solution. It is nothing more than a statement that an ambiguity problem has been recognized. As OCL claims to be a precise language, an alternative approach has definitely to be preferred.

5 Conclusion

This paper presents different proposals to eliminate an ambiguity problem in the current version of OCL. Two of these proposals seem to resolve the problem in an adequate way and should be further discussed at the workshop: Either a second type of actual parameter list is introduced to the OCL grammar or an identifying context must be specified for each actual parameter. Leaving the problem to the modelers cannot be regarded as a final solution, but at least some advice could be given to the OCL specification document until a formal solution is found.

Literature

[1] Klasse Objecten. *The Draft 1.4 OCL Grammar, Version 0.1b*, June 2000. http://www.klasse.nl/ocl/ocl-grammar-01b.pdf

[2] Klasse Objecten. *UML 1.4 RTF: OCL Issues - Changes from 1.3 to 1.4*, page 30, March 2000. http://www.klasse.nl/ocl/ocl-issues.pdf

[3] Object Management Group. *UML Unified Modelling Language Specification, Version 1.3*, March 2000. ftp://ftp.omg.org/pub/docs/formal/00-03-01.pdf

[4] J. Warmer and A. Kleppe. *The Object Constraint Language. Precise Modeling with UML*, Object Technology Series, Addison-Wesley, 1999.