On the Support of Qualified Associations in OCL

Lars Hamann[1], Martin Gogolla[2], and Nisha Desai[2]

[1] hagebau IT GmbH, Soltau, Germany
lars.hamann@hagebau.com
[2] University of Bremen, Bremen, Germany
{gogolla|nisha}@informatik.uni-bremen.de

Abstract. Associations are the glue that keeps items in object-oriented systems together. In UML qualified associations are used to express additional information about access to connected objects. From an implementation point of view, qualified associations can be used to define access to multi-dimensional data structures like arrays. From a conceptual point of view, they can be looked at as being ternary associations with particular OCL constraints expressing the multiplicities. Qualified associations are only partly described in the OCL standard, for example, it is not clear how to access all qualifier values for a qualified object. This paper takes up such questions and makes proposals for the support of qualified associations in the context of OCL.

Keywords: UML, OCL, qualified association, navigation.

1 Introduction

In object-oriented modeling languages like UML [14], structural class modeling is supported by query and constraint expressions formulated in OCL [16, 1]. Associations [15] play a central role in structural modeling because they provide the glue that keeps the system together. Among the many facets that associations can display (like association class, aggregation, composition, or derived association), qualified associations are among the rarely discussed topics.

Qualified associations are typically used to indicate fast access when navigating from one class to the other: "An index in a database ... is properly modeled as a qualifier" [14]. The index attributes in the database correspond to the qualifier attributes in the model. And qualified associations are employed to partition the links interpreting the association according to instances possessing particular properties.

One can view qualified associations from different perspectives. Qualified associations can be looked at from the implementation point of view. Then they are discussed in connection with arrays. If one looks at them from a conceptual modeling language point of view, they are discussed in connection with modeling features like association classes or ternary associations. They may also be transformed into such simpler modeling language elements.

Qualified associations enrich an association end with information about access to related objects. They must be studied with care in particular with regard

to the questions how many qualified objects and qualifier values can be connected to one or many target objects "on the other side of the association". Thus whether to consider sets or bags for typing in this context becomes crucial.

In current OCL [12, 13], qualified associations are somewhat underrepresented resp. underspecified. One cannot define constraints on qualifier attributes [4] unless one models information in redundant form, but nevertheless there is no way in OCL to access qualifier attributes in the sense that one can build expressions returning a qualifier value. It is only possible to employ qualifier attributes for accessing the target objects in the qualified association.

Let us explain our ideas with an example. In Fig. 1 the qualified association between the classes University and Student allows the modeler to access a unique Student, if a University object and a matriculation number is given.

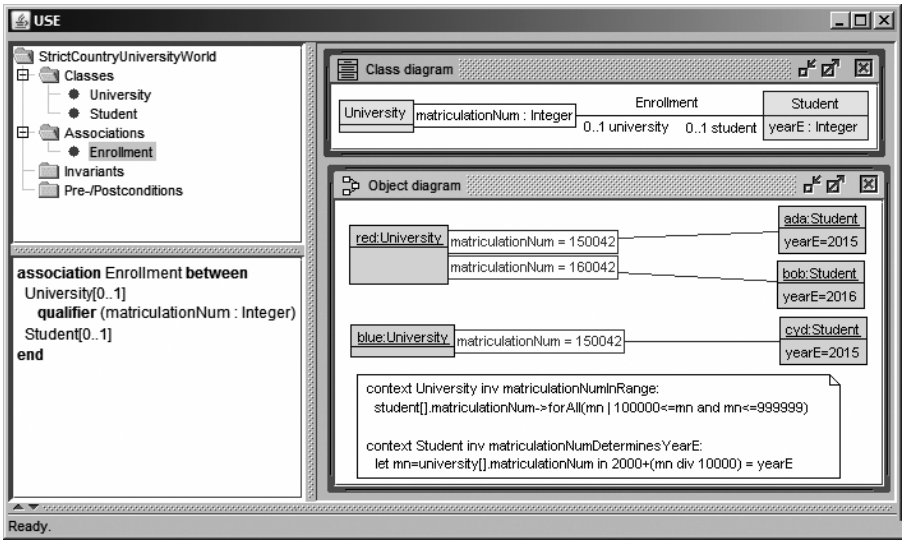


Fig. 1. Example for currently unsupported OCL constraints for qualified associations

The matriculation numbers are supposed to have a particular format consisting of the last two digits YY of the year of enrollment of the student followed by a running number NNNN within the enrollment year consisting of four digits: YYNNNN. In order to formally establish the requirements, two OCL invariants have been formulated in the comment node within the object diagram. These invariants cannot be formulated with current OCL, because an access to qualifier attribute values is needed which is currently not possible. In the first expression, self.student[].matriculationNum delivers for self:University all matriculation numbers within that university. In the second expression, self.university[].matriculationNum returns for self:Student the ma-

triculation number for that Student object. Details of the bracket notation will be explained below.

The rest of the paper is structured as follows. Section 2 discusses qualified associations from two points of view, namely from an implementation point of view and from a conceptual point of view. Section 3 puts forward our proposals for typing OCL expressions in the context of qualified associations and for accessing the values of qualifier attributes with OCL expressions. Section 4 discusses related work. The final section concludes and sketches future work.

2 Qualified Associations

In this section qualified associations are described using two different views. At first, an implementation oriented explanation is given. Afterwards, they are explained using a conceptual view.

Qualified associations can be used to model technical details about the implementation of associations. They enrich general associations with information about which data is required to access linked objects, i. e., the source object together with the qualifier values defines the result of a navigation. Informally, they also indicate that fast access to the linked objects is desired. This makes them especially useful for models that are close to the implementation, like platform specific models or models for runtime verification [9].

Figure 2 shows a qualified association with two qualifiers of type integer that closely matches the implementation in Java as a two-dimensional array given above the figure. The given example defines the structure of a game map similar to a chess board. The map consists of tiles that are uniquely defined by their (x,y)-coordinates. Please note, that the UML model contains additional information in contrast to the shown implementation only naming the two dimensions of the array.

```
1 public class Map {
2   private Tile[][] tiles;
3 }
```

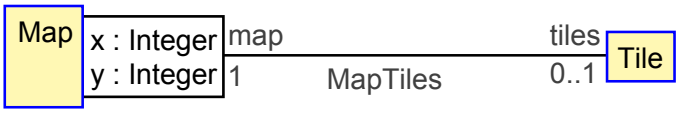


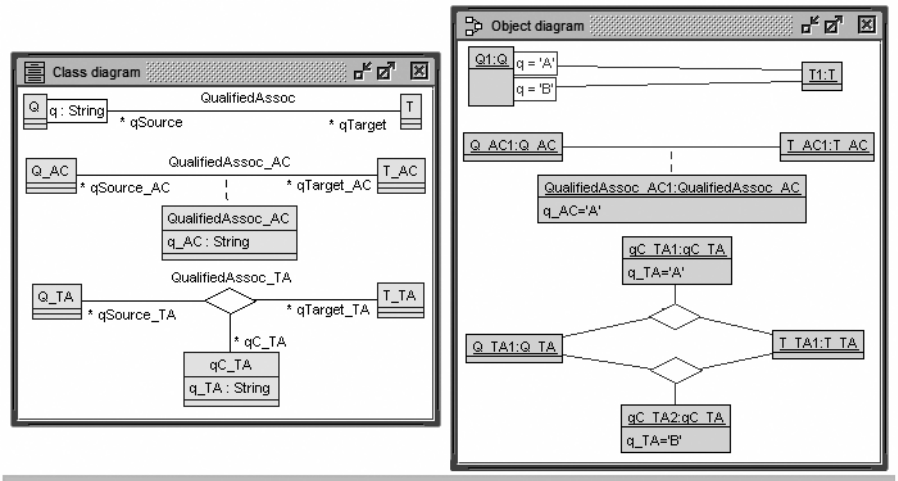
Fig. 2. A qualified association

Qualified associations are not only useful to represent arrays. They can also be used to overcome a drawback in OCL, namely the lack of a collection type to express map-like data structures.

Apart from looking at a qualified association from the implementation point of view, one can also look at it from the perspective of other UML and OCL modeling features that express the same characteristics. Figure 3 shows in the lower part how a qualified association can be understood as a ternary association.

With an additional OCL constraint one can express the qualified association multiplicity. The qualifier becomes an attribute of an additional class in the ternary association. The constraint basically states that for all qualified objects `self` and for all possible qualifier values `qv`, the size of the set of target objects being also linked to the same qualified object `self` and linked to the qualifier value `qv` is bound by the multiplicities `LL` and `HH` (for an `LL..HH` multiplicity range).

A transformation from a qualified association into an association class is implicitly suggested in [14] and has been proposed in [8]. However the current transformation into a ternary association captures more cases. The middle parts of Fig. 3 sketch a transformation into an association class, but they also directly indicate that a situation where one qualified object being connected with two different qualifier values cannot be expressed, if the association class is equipped with a set of links. A link between the qualified object and the target object can only be equipped with one qualifier value, but not with two.




```
context self:Q_TA inv QualifiedAssocMultiplicity:
  let LL=0 in let HH=1 in
  qC_TA.allInstances().q_TA->forAll( qv |
    let targetSameSelfSameQV_Size=T_TA.allInstances()->
      select( t | t.qSource_TA->includes(self) and
                  t.qC_TA.q_TA->includes(qv) )->size in
    LL<=targetSameSelfSameQV_Size and targetSameSelfSameQV_Size<=HH )
```

Fig. 3. Qualified associations transformed into associations

As far as we know, the need for an interpretation of qualified associations in terms of ternary associations has not been discussed so far. Further down in Fig. 7 and 8 we will show how the two introductory examples for qualified associations are formulated in form of ternary associations.

The translation into ternary association including standard OCL constraints is not only an intellectual exercise. For treating advanced UML modeling features in the context of validation and verification we translate advanced modeling concepts into a so-called base model [10]. Our underlying proving engine basically only handles plain associations and not advanced concepts as qualified associations. Thus, in that context we explicitly need this transformation.

3 Qualified Associations in OCL

The navigation over qualified associations is only sketched in the current OCL specification [13, p. 22,144]. For example, it lacks information about the backward navigation from the target of a qualified association. Consider again the example of a qualified association between a map and its tiles shown in Fig. 2. The OCL specification [13, p. 22] describes the results of the first two navigation expressions shown in Listing 1.1. However, no information about the result type of the third navigation expression is given. The first and the second expression are shown as they would be typed in the modeling tool USE [7].

Listing 1.1. Types of navigation expressions for qualified associations

```
1 aMap.tiles[1,1] : Tile (with aMap : Map,
2   i.e., aMap ia an expression of class Map)
3 aMap.tiles : Set(Tile)
4 aTile.map : UNCLEAR (with aTile : Tile)
```

Like the authors of [4], we propose to extend the applicability of qualified associations and especially of the qualifier in OCL as described next. In detail, we propose to

1. clearly define types and semantics of navigations for all relevant combinations of multiplicities, and
2. add support for the access to the values of qualifiers.

3.1 Navigation

On the type level, a navigation expression including qualifier values (line 1 in the above listing) behaves like a normal navigation using an unqualified association. If the upper bound of the multiplicity is one, the result type is the class at the target end. Otherwise a collection conforming to the end properties (ordered, unique, etc.) of the type at the target end is returned. The linked objects are determined by the link set of the association and the qualifier value.

If no qualifier values are provided, which is also a valid navigation [12, p. 21], the result type of such an expression is a set of objects of the target class

as it is informally stated by the OCL specification. A user can only deduct the result type by examining the provided example model in [12]. Taking the source multiplicity into account, the type `Set(Tile)` as the result type of the navigation expression is valid because a single tile can only be linked to one pair of map and an (x,y)-coordinate. Therefore, a map can only be linked once to a given tile making the result of the navigation set-valued. If the upper bound of the source multiplicity is greater than one, this statement does not hold anymore. Since the upper bound now specifies that a single tile can be linked with multiple pairs of map objects and (x,y)-coordinates, it is not guaranteed, that a tile is used only once in a single map object. To avoid a loss of information, the result of the navigation should lead to a bag.

For the opposite navigation over a qualified association, i. e., from the target end to the qualified end, the OCL specification contains no information. Following the semantics of non-qualified navigations, the result type of the navigation depends on the upper bound of the multiplicity of the target end. If it is one, the result type is the type of the class at this end. The default[3] type of a navigation to an association end with a multiplicity greater than one is defined as a set of linked objects at this end. Navigation over qualified associations cannot follow this definition if no loss of information is pursued. Given a multiplicity greater than one, the result type should be a bag of the type of the class at this end, because a target object can be linked by multiple pairs consisting of qualified object and qualifier. Figure 4 shows such a situation where a tile is connected twice to a single map by using different qualifier values. The navigation `tile1.map` should retrieve all available information and therefore it should return `Bag{map2,map1,map1}`.

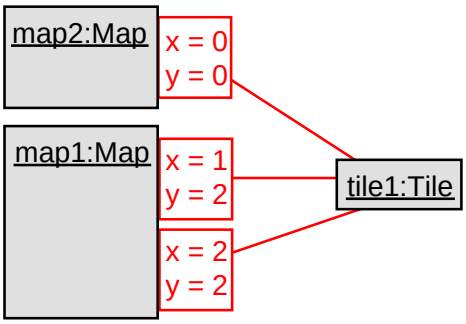
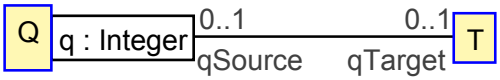


Fig. 4. An example state for a qualified association

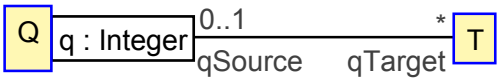
All four relevant combinations of multiplicity upper bounds are shown in Fig. 5. For each combination the three navigations and their type as we propose

[3] In this context default means the association is marked as unique.

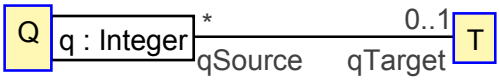
are shown below the example. Further, the type of the expression when following the OCL specification is given.




```
:aQ.qTarget[1]  -> T       // OCL spec.: T
:aQ.qTarget     -> Bag(T)  // OCL spec.: Set(T)
:aT.qSource     -> Q       // OCL spec.: UNCLEAR
```




```
:aQ.qTarget[1]  -> Set(T)  // OCL spec.: Set(T)
:aQ.qTarget     -> Bag(T)  // OCL spec.: Set(T)
:aT.qSource     -> Q       // OCL spec.: UNCLEAR
```




```
:aQ.qTarget[1]  -> T       // OCL spec.: T
:aQ.qTarget     -> Bag(T)  // OCL spec.: Set(T)
:aT.qSource     -> Bag(Q)  // OCL spec.: UNCLEAR
```


```
:aQ.qTarget[1]  -> Set(T)  // OCL spec.: Set(T)
:aQ.qTarget     -> Bag(T)  // OCL spec.: Set(T)
:aT.qSource     -> Bag(Q)  // OCL spec.: UNCLEAR
```

Fig. 5. Navigation over qualified associations

3.2 Qualifier Values in OCL Expressions

Our second proposal is to make qualifier values to first class citizens in OCL as it also proposed in [4]. In OCL 2.4 qualifier values cannot be queried. Therefore, some other information store, like for example attributes, are required to define constraints on them. This complicates the definition of model constraints or

leads to unnecessary duplications. Consider our example using maps and tiles. A possible extension would be to add constraints to the class `Tile` that require information about the position of the tile. One could add the attributes `x:Integer` and `y:Integer` to the class, but this does not provide any new information to the model. It is some kind of a workaround.

Therefore, we propose an extension to OCL, similar to the one presented in [4]. In contrast to our work, the authors propose to add a navigation expression to access all (key,value)-pairs of a qualified association. For this, they suggest to use empty brackets []. For the result the authors introduce a new collection type `Map`. While we follow the syntax, we argue that a map is inadequate since multiple qualifiers are not handled. To be able to handle qualified associations with multiple qualifiers, we propose to return the linked object together with the qualifier values as an OCL tuple. The tuple parts are built by the name of the target end and the names of the qualifier. The bracket notation can be used on both sides of the qualified association.

The following listing shows the result of such navigation expressions when evaluated using the system state shown in Fig. 4:

```
?map1.tiles[]
-> Set{Tuple{tiles=Tile1, x=1, y=2},
       Tuple{tiles=Tile1, x=2, y=2}}
  :Set(Tuple(tiles:Tile, x:Integer, y:Integer))
?map1.tiles[].y
-> Bag{2, 2}
  :Bag(Integer)
?tile1.map[]
-> Set{Tuple{map=map2, x=0, y=0},
       Tuple{map=map1, x=1, y=2},
       Tuple{map=map1, x=2, y=2}}
  :Set(Tuple(map:Map, x:Integer, y:Integer))
```

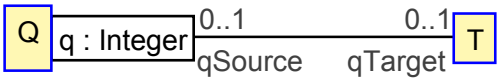
The resulting set of tuples can now be used as any other set in OCL to define constraints or to query the system state.

For the general case of bracket navigation, the four relevant combinations of multiplicity upper bounds are also shown in Fig. 6. There, for each combination the two bracket navigations and their result type are shown. Two examples for bracket navigations have already been given in Fig. 1.

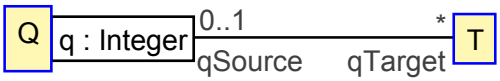
3.3 Further examples

Figures 7 and 8 show how the two introductory examples are represented as ternary associations together with an appropriate OCL constraint. Please note that in the case of two or more qualifier attributes the stated constraint schema has to be slightly extended to capture more qualifiers. However, the basic underlying structure of the constraints does not change.

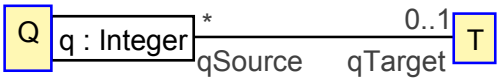
Figure 9 shows an example for using qualified association with a multiplicity of more than one on the target side. We have not found such an example in the




```
:aQ.qTarget[] -> Set(Tuple(qTarget:T,q:Integer))
:aT.qSource[] -> Tuple(qSource:Q,q:Integer)
```




```
:aQ.qTarget[] -> Set(Tuple(qTarget:T,q:Integer))
:aT.qSource[] -> Tuple(qSource:Q,q:Integer)
```




```
:aQ.qTarget[] -> Set(Tuple(qTarget:T,q:Integer))
:aT.qSource[] -> Set(Tuple(qSource:Q,q:Integer))
```


```
:aQ.qTarget[] -> Set(Tuple(qTarget:T,q:Integer))
:aT.qSource[] -> Set(Tuple(qSource:Q,q:Integer))
```

Fig. 6. Navigation over qualified associations with bracket notation []

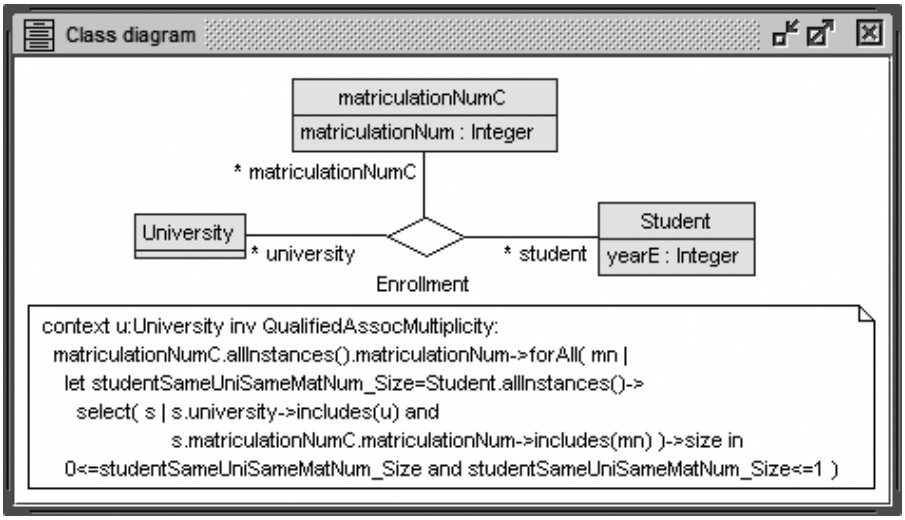


Fig. 7. University example transformed into ternary associations

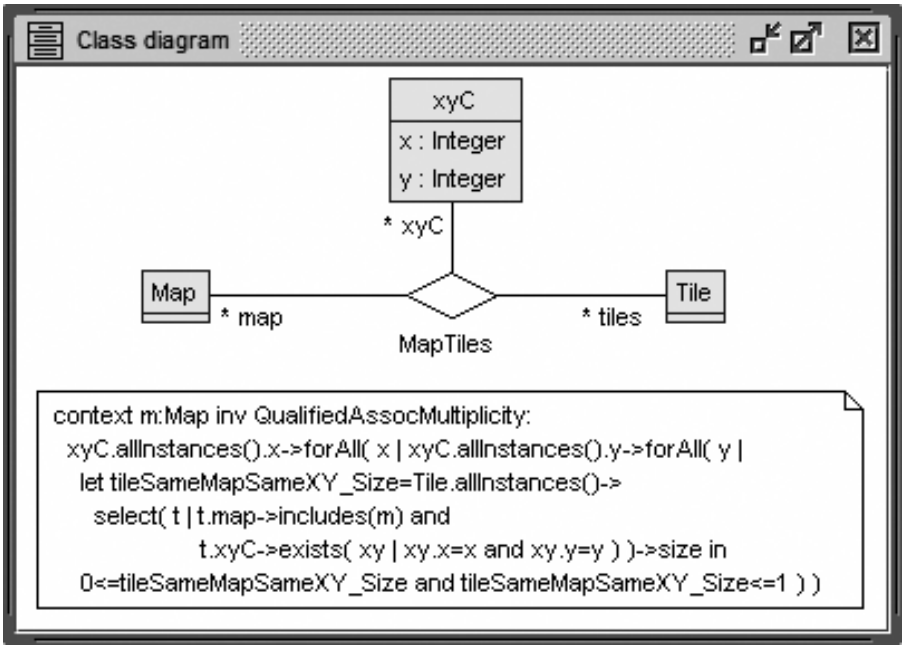


Fig. 8. MapTiles example transformed into ternary associations

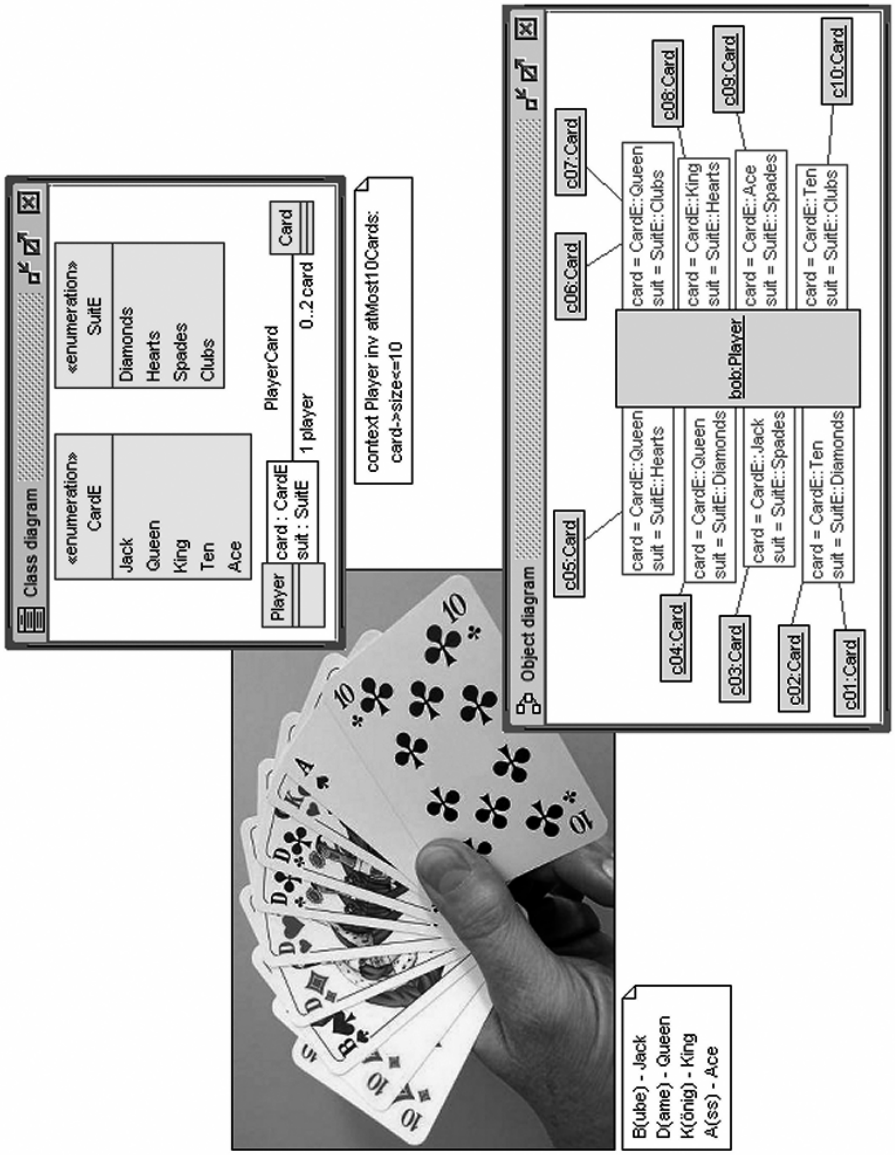


Fig. 9. Example for a qualified association with set-valued target multiplicity

literature. The example is part of a model for a German card game called "Doppelkopf". In that game each card of an ordinary card collection is present twice, as shown in the object diagram in which the Diamonds Ten and the Clubs Queen are twice present on the hand of a card player. Please note that there are ten card objects, but only eight qualifier values; the qualifier values (Diamonds,Ten) and (Clubs,Queen) are both connected to two card objects.

4 Related work

As already stated, the semantics of qualified associations by transforming them into equivalent constructs in UML using additional constraints were already discussed in [5, 8]. In this paper we extended this transformation by considering ternary associations. The link between UML associations including qualified associations and their implementation is examined in [2, 3].

In [11] the formal semantics of associations and association ends are discussed in detail. The author shows several arising inconsistencies by using constraints like *unique*. While association classes are covered, qualified associations are not. Work on the mapping between UML unqualified associations and their Java implementation has been published in [6].

The extension to OCL proposed in [4] is similar to our proposal, but works only for qualified associations with one qualifier. As we have shown, our modified and extended version of that proposal also supports multiple qualifiers.

5 Conclusion

This paper has discussed support of qualified associations in OCL. In particular we have clarified typing issues (sets vs. bags) for qualified associations distinguishing between different stated multiplicities. We have made a proposal for accessing qualifier values which is not possible employing the current OCL standard features. In particular constraints formulating dependencies between classifier values and other modeling items now become possible. We have clarified and extended the options from the current OCL standard.

Future work includes completing and polishing the implementation of our proposal in the tool USE. Discussions with the OCL community may lead to further requirements concerning qualified associations. Extended OCL support for other association-like relationships like aggregation and composition might be useful and worth to be studied as well. For example, in the context of composition one might introduce a particular OCL expression for navigating to the aggregate object along the black diamonds without having to explicitly mention association end names.

References

1. Cabot, J., Gogolla, M.: Object Constraint Language (OCL): A Definitive Guide. In: Bernardo, M., Cortellessa, V., Pierantonio, A. (eds.) Proc. 12th Int. School For-

mal Methods for the Design of Computer, Communication and Software Systems: Model-Driven Engineering. pp. 58–90. Springer, Berlin, LNCS 7320 (2012)
2. Diskin, Z., Dingel, J.: Mappings, maps and tables: Towards formal semantics for associations in UML2. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 230–244. Springer (2006)
3. Diskin, Z., Easterbrook, S., Dingel, J.: Engineering Associations: From Models to Code and Back through Semantics. In: Paige, R.F., Meyer, B., Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M.J., Szyperski, C. (eds.) Objects, Components, Models and Patterns, LNBP, vol. 11, pp. 336–355. Springer Berlin Heidelberg (2008)
4. Dove, A., Barua, A., Cheon, Y.: Extending OCL to Better Express UML Qualified Associations. Tech. rep., Department of Computer Science, University of Texas at El Paso (2014)
5. Flake, S.: Eliminating Qualifier and Association Class Ambiguities from OCL. UML 2.0: The Future of the UML Object Constraint Language (OCL), UML Conference Workshop (2000)
6. Gessenharter, D.: Mapping the UML2 Semantics of Associations to a Java Code Generation Model. In: Czarnecki, K., Ober, I., Bruel, J.M., Uhl, A., Völter, M. (eds.) Model Driven Engineering Languages and Systems, LNCS, vol. 5301, pp. 813–827. Springer Berlin / Heidelberg (2008)
7. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming 69, 27–34 (2007)
8. Gogolla, M., Richters, M.: Expressing UML Class Diagrams Properties with OCL. In: Clark, T., Warmer, J. (eds.) Advances in Object Modelling with the OCL, pp. 86–115. Springer, Berlin, LNCS 2263 (2001)
9. Hamann, L.: On formalizing UML and OCL features and their employment to runtime verification. Ph.D. thesis, University of Bremen (2015), http://elib.suub.uni-bremen.de/edocs/00104250-1.pdf
10. Hilken, F., Niemann, P., Gogolla, M., Wille, R.: From UML/OCL to Base Models: Transformation Concepts for Generic Validation and Verification. In: Kolovos, D., Wimmer, M. (eds.) Proc. 8th Int. Conf. Model Transformation (ICMT 2015). pp. 1–17. Springer, LNCS 9152 (2015)
11. Milicev, D.: On the Semantics of Associations and Association Ends in UML. Software Engineering, IEEE Transactions on 33(4), 238 –251 (april 2007)
12. Object Constraint Language 2.3.1. Object Management Group (OMG) (Jan 2012), http://www.omg.org/spec/OCL/2.3.1/
13. Object Constraint Language 2.4. Object Management Group (OMG) (Feb 2014), http://www.omg.org/spec/OCL/2.4/
14. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, 2nd Edition (2004)
15. Stevens, P.: On the interpretation of binary associations in the Unified Modelling Language. Software and System Modeling 1(1), 68–79 (2002)
16. Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley (2003), 2nd Edition