Synchronizing Refactored UML Class Diagrams and OCL Constraints

Slaviša Marković and Thomas Baar
École Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
CH-1015 Lausanne, Switzerland
Email: {slavisa.markovic, thomas.baar}@epfl.ch

***Abstract*—UML class diagrams are usually annotated with OCL expressions that constrain their possible instantiation. In our work we have investigated how OCL annotations can be automatically updated each time the underlying diagram is refactored. All our refactoring rules are formally specified using a QVT-based graphical formalism and have been implemented in our tool ROCLET.**

I. Refactoring Class Diagrams

In this section we give a motivation for performing UML/OCL refactorings and show on an example, how OCL constraints have to be treated when the underlying UML class diagram changes. Note that our approach does *not* aim to improve the structure of OCL expressions in order to get rid of OCL smells (see [1]). We are just concerned about smells in UML class diagrams, how to eliminate these smells by class diagram refactorings, and how to keep the annotated OCL constraints in sync with these changes.

Figure 1 shows the application of refactoring *MoveAttribute* on a class diagram annotated with one OCL invariant. The refactoring moves attribute `telephone` from class `Person` to class `Info`. In order to preserve the syntactical correctness of annotated constraints, it is necessary to rewrite all navigation expressions of form *exp*`.telephone` by *exp*`.info.telephone`.

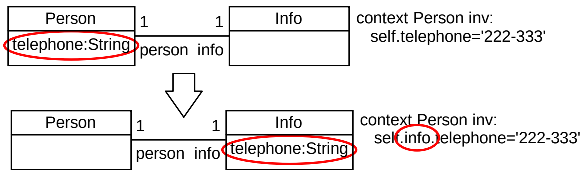

Fig. 1. *MoveAttribute* Example

The only preconditions for this refactoring are (1) that the attribute is moved over an association with multiplicities 1-1 and (2) that in the destination class (`Info`), or in any of its descendants and ancestors, there is no attribute with the same name as the name of the moved attribute (`telephone`).

There are other refactoring rules, which do not influence annotated OCL constraints but whose applicability depends on the absence of OCL expressions of a certain type. One example is the rule *PushDownAttribute*; Fig. 2 shows an application where attribute `color` is pushed down from class `Vehicle` to class `Car`. This refactoring is only possible if for all occurring expressions *exp*`.color` the type of subexpression *exp* conforms to destination class `Car`.

TABLE I
OVERVIEW OF UML/OCL REFACTORING RULES

Refactoring rules	**Influence on OCL**	**Precondition**
RenameClass	No*	UML
RenameAttribute	No*	UML
RenameOperation	No*	UML
RenameAssociationEnd	No*	UML
PullUpAttribute	No	UML
PullUpOperation	No	UML/OCL
PullUpAssociationEnd	No	UML/OCL
PushDownAttribute	No	UML/OCL
PushDownOperation	No	UML/OCL
PushDownAssociationEnd	No	UML/OCL
ExtractClass	No	UML
ExtractSuperclass	No	UML
MoveAttribute	Yes	UML
MoveOperation	Yes	UML
MoveAssociationEnd	Yes	UML

*–*Rename* refactorings influence textual notation of OCL constraints but not their metamodel representation

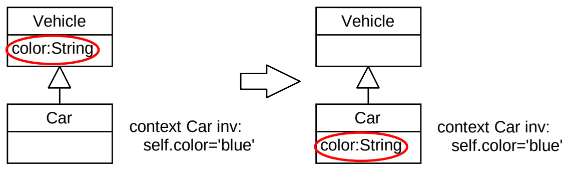

Fig. 2. *PushDownAttribute* Example

In [2], we have investigated and formalized a catalog of class diagram refactorings together with necessary changes of OCL constraints. Table I gives an overview of refactorings that can be applied on a class diagram, together with information whether the refactoring influences OCL constraints, and which part of the UML/OCL model is checked by the refactoring's application condition.

II. Model Transformations

UML class diagrams and their OCL constraints can be seen as models (i.e. instances of corresponding metamodels). The refactoring of UML/OCL models is a special type of model transformation and can, thus, be specified by the OMG standard QVT (Query/View/Transformation).

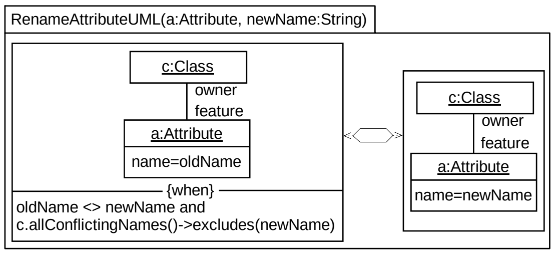

Fig. 3. QVT Formalization of *RenameAttribute* Refactoring

The formalization of the refactoring (*RenameAttribute*) is shown as a QVT rule in Fig. 3. QVT rules consist of basically two patterns (LHS, RHS). When applying the rule, occurrences of LHS are searched for in the non-refactored model that we want to change. If an occurrence is found, it is substituted with the corresponding instantiation of RHS. Additional constraints specified in the "when" clause specify formally the application conditions for the refactoring rule (ignoring them could result in syntactically invalid target models). For more information on the formalization of refactorings, we refer the interested reader to [2]. The refactoring rule *RenameAttribute* does not have an influence on attached OCL constraints. More complicated rules that have an influence (e.g. *MoveAttribute*), are formalized by two QVT rules; one describing the changes in the class diagram and a second for updating the OCL (see [2] for details).

III. Lessons Learned

A model refactoring is usually defined as a model transformation for which source and target model are instances of the same metamodel. During our work on implementing QVT-specified refactoring rules we have noticed that it is sometimes useful to relax this definition and to allow source and target model to have different metamodels.

A. Syntax Preservation

Refactoring rules should be syntax-preserving; i.e. syntactically correct source models should always be mapped to syntactically correct target models. However, syntax preservation is sometimes technically difficult to achieve, especially, if the metamodel contains hundreds of well-formedness rules.

Syntax preservation becomes easier to handle when refactoring is seen as a two-step process: (1) the source model is transformed to an intermediate model, which is an instance of a different metamodel; (2) from the intermediate model the final target model is recovered by a second transformation. In case of UML/OCL refactorings, the intermediate metamodel could represent OCL constraints as text and the refactoring rules just have to "produce" text in order to represent synchronized OCL constraints. The second recovery step would then parse the produced text as OCL constraints and create an instance of the original UML/OCL metamodel. Another possibility for an intermediate metamodel could be to use the original UML/OCL metamodel, but without any of its derived model elements. In this case, the only task of the recovery step would be to complete the intermediate model to an instance of the original UML/OCL metamodel by adding the (so far missing) derived model elements.

B. Behavior Preservation

In case of UML class diagram refactorings, the definition of behavior preservation in traditional program refactoring as "same inputs lead to the same output" is not applicable because class diagrams represent only the static structure of a system.

Our criterion for behavior preservation is based on the evaluation of OCL constraints in a system snapshot. In [3], we propose to call UML/OCL refactorings *behavior preserving* if the evaluation of a non-refactored OCL constraint on a valid instance of a non-refactored UML class diagram yields always the same result as the evaluation of the refactored OCL constraint on the corresponding instance of the refactored UML class diagram.

Contrary to some authors, like [4], we allow object diagrams also to be refactored. We believe that our definition of semantic correctness gives more freedom in performing refactorings and allows wider spectrum of refactoring rules to be applied on a UML class diagram.

IV. Conclusions

In this paper we have presented our approach of refactoring UML class diagrams annotated with OCL constraints. All refactorings that can be applied on class diagrams are specified as model transformation rules and implemented in our RoclET tool [5].

Moreover, an overview of lessons learned during the process of formalization and implementation is given. We think that the technique to handle refactorings as a 2-step process can help to simplify the refactoring of many other software artifacts as well.

References

[1] Alexandre Correa and Cláudia Werner. Applying refactoring techniques to UML/OCL. In *UML 2004*, volume 3273 of *LNCS*, pages 173–187. Springer, 2004.

[2] Slaviša Marković and Thomas Baar. Refactoring OCL annotated UML class diagrams. *Software and Systems Modeling (SoSym)*, 2007. In press. Online available under DOI 10.1007/s10270-007-0056-x.

[3] Thomas Baar and Slaviša Marković. A graphical approach to prove the semantic preservation of UML/OCL refactoring rules. In *PSI 2006*, volume 4378 of *LNCS*, pages 70–83. Springer, 2007.

[4] Rohit Gheyi, Tiago Massoni, and Paulo Borba. A static semantics for alloy and its impact in refactorings. *Elsevier's Electronic Notes in Theoretical Computer Science (To appear)*, 2006.

[5] RoclET homepage. http: //www.roclet.org, 2007.