UML and OCL Transformation Model Analysis: Checking Invariant Independence

Martin Gogolla, Frank Hilken

Database Systems Group, University of Bremen, Germany
{gogolla|fhilken}@informatik.uni-bremen.de

Abstract. This paper discusses a case study for showing invariant independence for a transformation model. The study is based on an approach that proposes to analyze UML and OCL models using a solver for relational logic. In the approach, UML and OCL models describe system structures formally with UML class diagrams and OCL class invariants. Test cases in form of object diagrams are constructed and employed for property inspection. With the approach one can prove model properties like model constraint independence for the structural model part. Thus important model properties can be analyzed on the modeling level without the need for implementing the model. All feedback given to the developer is stated in terms of the used modeling language, UML and OCL.

1 Introduction

Models and model transformations are regarded as essential cornerstones for Model-Driven Engineering (MDE) [6, 19, 12]. Quality improvement techniques like validation and verification on the modeling level are central for the success of MDE, in particular when they are performed in combination with testing on the basis of solver engines. Thus testing and validation approaches for models and model transformations are obtaining more and more attention (see [4, 3]).

The starting point for our work is a UML class model that is completed by OCL invariants. Analyzing that model then means to verify properties like consistency or constraint independence. Our approach proposes to check such properties by applying a so-called model validator in the tool USE (Uml-based Specification Environment, see [13] for basic functionality of an early version) that searches for test cases in form of object diagrams. We apply our technique here in an exemplary way to model transformations in form of transformation models [5] and present a case study, whereas the basic approach has already been put forward in [14, 15], it has not been demonstrated to work for invariant independence for transformation models.

The rest of this paper is structured as follows. Section 2 gives an overview on the verification options of the approach. Section 3 describes the case study in form of a transformation model. Section 4 handles one particular verification option, namely invariant independence that has not been demonstrated within

the current approach before. Section 5 points to related work. Section 6 closes the paper with a conclusion and future work. More details about the transformation model, configurations and execution have been presented in [14, Additional Material].

2 Overview on Verification Options within the Approach

Figure 1 presents an overview on different options that the USE model validator offers. The starting point is that a developer has prepared a UML and OCL model and wants to verify properties of the model. The developer must do so by working out an appropriate configuration for the model validator, i.e., determine possible class and association populations and possible attribute and datatype values. Then a developer has the following options.

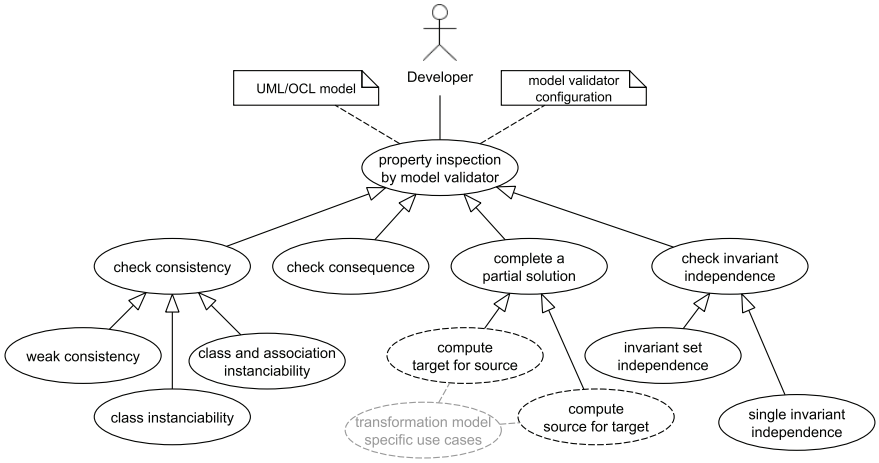

Fig. 1. Use cases for verification options within the USE model validator.

Consistency: The consistency of the model can be checked, i.e., it is analyzed whether the model can be instantiated. This shows that the model constraints are not contradictory. Details are available in [14].

Consequences: Consequences of the model can be inspected. For this purpose, the negation of the suspected consequence together with the stated constraints are handed to the model validator and a counterexample is searched for in the finite search space determined by the configuration. If none is found, the assumed consequence is regarded as valid. Details are available in [14].

Partial solution completion: A partial solution of the model may be given to the model validator who tries to complete the partial solution. In terms of our running example, a transformation model, this can be utilized for explicitly computing for a source instance a target instance and vice versa. Transformation properties like injectivity can be checked in this way. Details are available in [15].

Invariant independence: A set of invariants can be tested for independence, i.e., it is checked that one invariant is not implied by other invariants, or in

other words that the considered invariant is independent from the other ones. This may be regarded as a kind of minimality property of the invariant set. One could also call it redundancy freeness. This property has not yet been elaborated in connection with the present USE model validator.

All feedback is given in terms of the language that the developer uses for describing models: UML and OCL. All results are presented in form of UML, not as an internal representation of the underlying solver engine. OCL can be employed to inspect the resulting object diagram.

3 Model Transformation Example

The running example in this paper is the well-known transformation between ER and relational database schemata. We study this transformation in form of a transformation model as introduced in transformation model is a descriptive

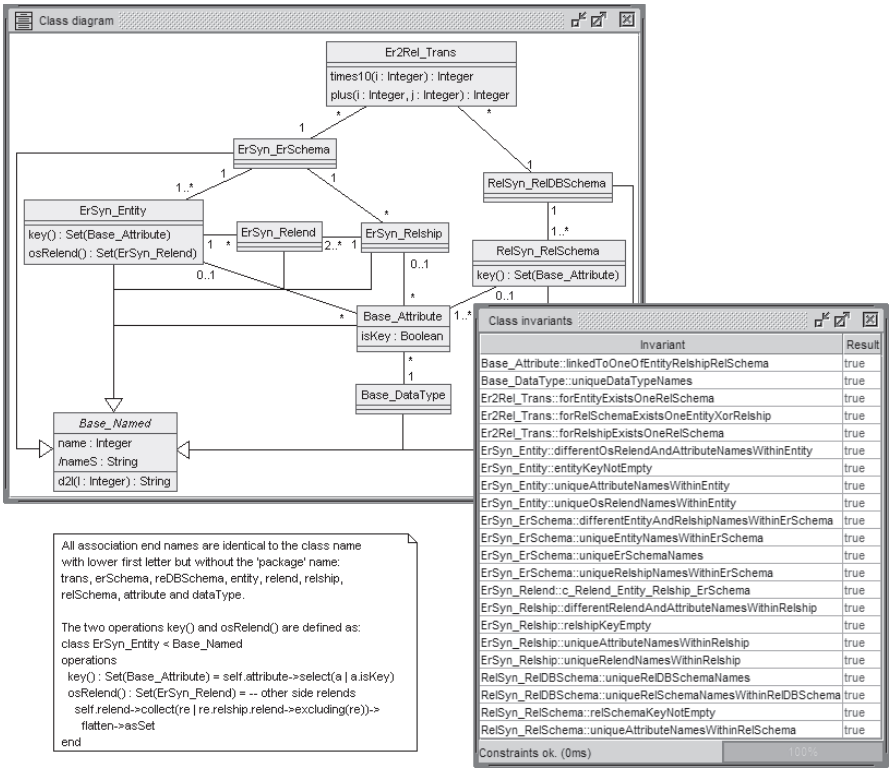

Fig. 2. Class diagram and invariants for example transformation model.

model where the relationship between source and target is purely characterized by the (source,target) model pairs determined by the transformation. A transformation model consists in our approach of a plain UML class diagram with restricting OCL invariants. Typically, there is an anchor class for the source

model, an anchor class for the target model, and a connecting class for the transformation. In the example there are 22, partly non-trivial OCL invariants for restricting the source metamodel, for the target metamodel, and for the transformation. In Fig. 2 the class diagram and the invariant names for the example are depicted.

The example transformation model has four packages: a base part with datatypes and attributes for concepts commonly employed in the ER and relational model; a part for ER schemata (`ErSchema`) with the concepts `Entity`, `Relship` (relationship), and `Relend` (relationship end); a part for relational database schemata (`RelDBSchema`) incorporating relational schemata (`RelSchema`); finally, a part for the transformation (`Trans`). discusses also the semantics. Therefore, some classes here are marked `RelSyn`). More details of the example can be found in [14, additional material].

4 Invariant Independence

One crucial property for a set of invariants is whether the invariants are independent from each other, i.e., a single invariant cannot be obtained as a deduction from other stated invariants. In typical models, the invariants may be independent or there may be intentional dependencies. In any case, it is an interesting question to identify a minimal set of independent invariants. For a transformation developer invariant independence means that the invariant set in the transformation model is minimal and that exactly these conditions have to be respected, for example, in a transformation implementation.

Fig. 3. Model validator configuration for constraint independence.

Our approach supports the developer in order to check a set of invariants for independence under a *single* configuration which is shown in Fig. 3. For performing this task, all invariants are considered sequentially; each single invariant is negated and handed over together with the other invariants and the stated con-

figuration to the model validator. Figure 4 shows the resulting protocol for the running example and also presents time stamps (format HH:MM:SS) allowing to deduce the taken amount of time to compute a respective counterexample. Thus the independence proof takes about 4.5 minutes. The complete set with 22 invariants can be shown to be independent. 22 object diagrams (test cases) have been constructed. In every object diagram exactly one invariant is not satisfied.

```
modelvalidator -invIndep invariantIndependenceConfig.properties

15:18:24 Base_Attribute::linkedToOneOfEntityRelshipRelSchema: Independent
15:18:37 Base_DataType::uniqueDataTypeNames: Independent
15:18:48 Er2Rel_Trans::forEntityExistsOneRelSchema: Independent
15:19:17 Er2Rel_Trans::forRelSchemaExistsOneEntityXorRelship: Independent
15:19:29 Er2Rel_Trans::forRelshipExistsOneRelSchema: Independent
15:19:40 ErSyn_Entity::differentOsRelendAndAttributeNamesWithinEntity: Indpendent
15:19:50 ErSyn_Entity::entityKeyNotEmpty: Independent
15:20:00 ErSyn_Entity::uniqueAttributeNamesWithinEntity: Independent
15:20:11 ErSyn_Entity::uniqueOsRelendNamesWithinEntity: Independent
15:20:21 ErSyn_ErSchema::differentEntityAndRelshipNamesWithinErSchema: Indpendent
15:20:34 ErSyn_ErSchema::uniqueEntityNamesWithinErSchema: Independent
15:20:45 ErSyn_ErSchema::uniqueErSchemaNames: Independent
15:20:55 ErSyn_ErSchema::uniqueRelshipNamesWithinErSchema: Independent
15:21:06 ErSyn_Relend::c_Relend_Entity_Relship_ErSchema: Independent
15:21:19 ErSyn_Relship::differentRelendAndAttributeNamesWithinRelship: Indpendent
15:21:30 ErSyn_Relship::relshipKeyEmpty: Independent
15:21:41 ErSyn_Relship::uniqueAttributeNamesWithinRelship: Independent
15:21:52 ErSyn_Relship::uniqueRelendNamesWithinRelship: Independent
15:22:05 RelSyn_RelDBSchema::uniqueRelDBSchemaNames: Independent
15:22:16 RelSyn_RelDBSchema::uniqueRelSchemaNamesWithinRelDBSchema: Independent
15:22:28 RelSyn_RelSchema::relSchemaKeyNotEmpty: Independent
15:22:39 RelSyn_RelSchema::uniqueAttributeNamesWithinRelSchema: Independent
```

Fig. 4. Command line protocol for invariant independence with indicated time stamps.

It is also to inspect the counterexample (object diagram) constructed for a single invariant. Figure 5 displays the counterexample for the invariant `forRelshipExistsOneRelSchema`. Primarily, it is denoted as an object diagram for the underlying metamodel; but it is also denoted in a domain specific way partly as an ER diagram and partly as a relational textual SQL schema. The model validator has constructed an ER schema with one entity and one relationship, but only the entity is reflected in the relational DB schema. This leads to the fact that exactly one invariant is not satisfied in the object diagram and proves the independence of `forRelshipExistsOneRelSchema`.

Invariant independence is based on the equivalence that is stated below in an exemplary way for the case of three invariants. The independence statement may be expressed or read as: Inv_B is independent of $\{Inv_A, Inv_C\}$.

$$\begin{aligned} & \exists\, od{:}ObjectDiagram\,(Inv_A \wedge \neg Inv_B \wedge Inv_C) \\ \Leftrightarrow\ & \neg \forall\, od{:}ObjectDiagram\,(Inv_A \wedge Inv_C \Rightarrow Inv_B) \end{aligned}$$

5 Related Work

Related approaches: Our contribution is based on Alloy [17] and Kodkod [21]. OCL has already been employed for testing and verification. Mutation testing for

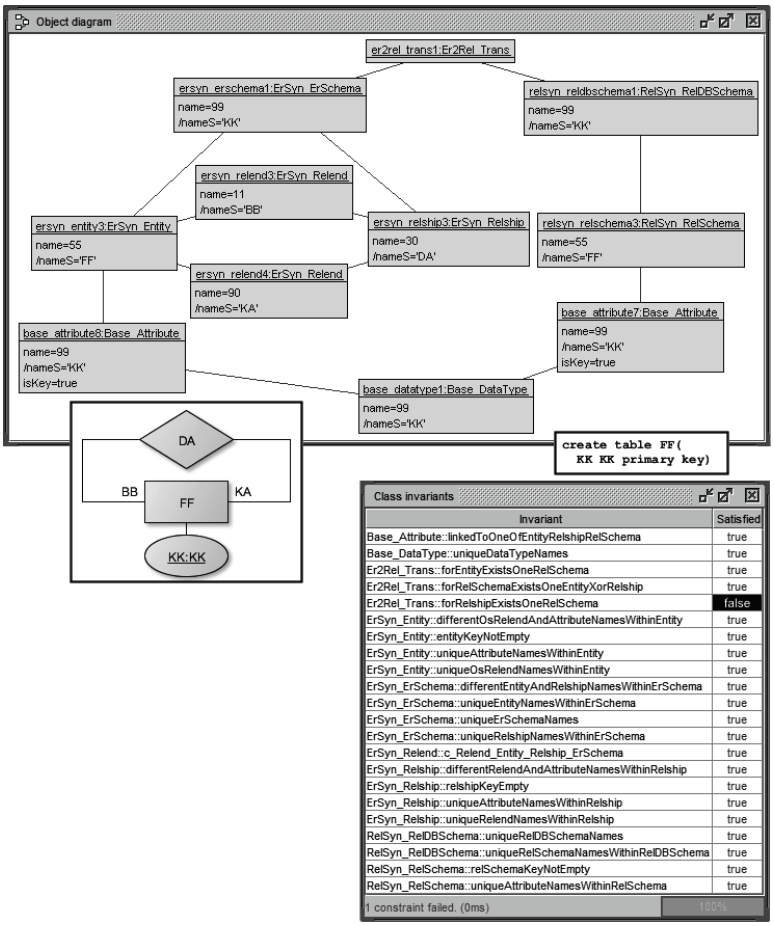

Fig. 5. Counterexample for independence of forRelshipExistsOneRelSchema.

OCL constraints has been considered in [1], and test data generation from OCL constraints was discussed in [2]. Validation of OCL and object-oriented models for particular areas like web forms [11], or conceptual DB schemas [20] has been studied as well. OCL property derivation based on interactive proving techniques is put forward in [7]. Transformation models using the same example as here, but focusing on refinement, have been studied in [8]. A general context of descriptive transformations employing UML and OCL is described in [9]. Different notions of consistency have also been proposed in [10]. In contrast to the mentioned works, our approach is the only one which supports full OCL and studies general properties like invariant independence by automatically constructing object diagrams.

Relationship to own work: The model validator is grounded on a translation of UML and OCL concepts into relational logic [18]. The same model transformation as here with focus on consistency and metamodel property preservation

has been studied in a workshop paper [14]. The completion technique was proposed in a workshop paper [15] as well. Results concerning the independence of invariants in the context of the model validator are original in this contribution. Additional material (complete model sources, configurations and additional examples) can be found in [14, additional material].

6 Conclusion and Future Work

The paper presented an approach for automatically checking UML and OCL model features. We have sketched how models could be checked for consistency, for consequences, and through the completion of partially specified object diagrams. We have shown how to analyze invariant independence, i.e., how to check the absence of redundancy in an invariant set.

Future work includes a number of topics. Apart from checking model properties, the model validator could be employed for systematic construction of test cases for UML models on the basis of an orthogonal test case characterization with OCL. Optimizations in the translation to the underlying relational logic could simplify the currently involved handling of the undefined value which is needed due to the fact that UML and OCL have more than two truth values. The handling of strings must be improved. Behavioral aspects must be integrated by transforming first state charts into pre- and postconditions and then pre- and postconditions into so-called filmstrip models that contain invariants only [16]. Last but not least, larger case studies must check the practicability of the approach.

References

1. Aichernig, B.K., Salas, P.A.P.: Test Case Generation by OCL Mutation and Constraint Solving. In: 2005 NASA / DoD Conference on Evolvable Hardware (EH 2005), 29 June - 1 July 2005, Washington, DC, USA, IEEE Computer Society (2005) 64–71
2. Ali, S., Iqbal, M.Z.Z., Arcuri, A., Briand, L.C.: Generating Test Data from OCL Constraints with Search Techniques. IEEE Trans. Software Eng. **39**(10) (2013) 1376–1402
3. Amrani, M., Lucio, L., Selim, G.M.K., Combemale, B., Dingel, J., Vangheluwe, H., Traon, Y.L., Cordy, J.R.: A Tridimensional Approach for Studying the Formal Verification of Model Transformations. In Antoniol, G., Bertolino, A., Labiche, Y., eds.: Proc. Workshops ICST, IEEE (2012) 921–928
4. Baudry, B., Ghosh, S., Fleurey, F., France, R.B., Traon, Y.L., Mottu, J.M.: Barriers to Systematic Model Transformation Testing. CACM **53**(6) (2010) 139–143
5. Bezivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model Transformations? Transformation Models! In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G., eds.: Proc. 9th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2006), Springer, Berlin, LNCS 4199 (2006)
6. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Morgan & Claypool (2012)

7. Brucker, A.D., Wolff, B.: Semantics, Calculi, and Analysis for Object-Oriented Specifications. Acta Inf. **46**(4) (2009) 255–284
8. Büttner, F., Egea, M., Guerra, E., de Lara, J.: Checking Model Transformation Refinement. In Duddy, K., Kappel, G., eds.: Proc. Inf. Conf. ICMT. LNCS 7909, Springer (2013) 158–173
9. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Verification and Validation of Declarative Model-To-Model Transformations through Invariants. Journal of Systems and Software **83**(2) (2010) 283–302
10. Cabot, J., Clarisó, R., Riera, D.: Verification of UML/OCL Class Diagrams using Constraint Programming. In: ICST Workshops, IEEE Computer Society (2008) 73–80
11. Escott, E., Strooper, P.A., King, P., Hayes, I.J.: Model-Driven Web Form Validation with UML and OCL. In Harth, A., Koch, N., eds.: Current Trends in Web Engineering - Workshops, Doctoral Symposium, and Tutorials, Held at ICWE 2011, Paphos, Cyprus, June 20-21, 2011. Revised Selected Papers. Volume 7059 of Lecture Notes in Computer Science., Springer (2011) 223–235
12. Frankel, D.: Model Driven Architecture: Applying MDA to Enterprise Computing. John Wiley & Sons, Inc., New York, NY, USA (2002)
13. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming **69** (2007) 27–34
14. Gogolla, M., Hamann, L., Hilken, F.: Checking Transformation Model Properties with a UML and OCL Model Validator. In: Proc. 3rd Int. STAF'2014 Workshop Verification of Model Transformations (VOLT'2014). (2014) CEUR Proceedings. Paper and Additional material: `http://www.db.informatik.uni-bremen.de/publications/intern/GHH2014VOLT.pdf`.
15. Gogolla, M., Hamann, L., Hilken, F.: On Static and Dynamic Analysis of UML and OCL Transformation Models. In: Proc. 3rd Int. MODELS'2014 Workshop Analysis of Model Transformations (AMT'2014). (2014) CEUR Proceedings. Paper: `http://www.db.informatik.uni-bremen.de/publications/intern/GHH2014AMT.pdf`.
16. Gogolla, M., Hamann, L., Hilken, F., Kuhlmann, M., France, R.B.: From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics. In Fill, H., Karagiannis, D., Reimer, U., eds.: Proc. Modellierung (MODELLIERUNG'2014), GI, LNI 225 (2014) 273–288
17. Jackson, D.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2006)
18. Kuhlmann, M., Gogolla, M.: From UML and OCL to Relational Logic and Back. In France, R., Kazmeier, J., Breu, R., Atkinson, C., eds.: Proc. 15th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2012), Springer, Berlin, LNCS 7590 (2012) 415–431
19. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, Boston (2004)
20. Queralt, A., Teniente, E.: Verification and Validation of UML Conceptual Schemas with OCL Constraints. ACM Trans. Softw. Eng. Methodol. **21**(2) (2012) 13
21. Torlak, E., Jackson, D.: Kodkod: A Relational Model Finder. In: Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2007). (2007) LNCS 4424, 632–647