Checking UML and OCL Model Behavior with Filmstripping and Classifying Terms

Martin Gogolla, Frank Hilken, Khanh-Hoang Doan, Nisha Desai

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

Abstract. This tool paper discusses how model behavior expressed in a UML and OCL model can be analysed with filmstrips and classifying terms in the tool USE. Classifying terms are a means for systematic construction of test cases. In the case of behavior models these test cases correspond to testing the model with different sequence diagrams. We explain how behavior analysis can be carried out in the tool. We discuss lessons learnt from the case study and how conceptual and technical support can be improved.

1 Introduction

Models are the cornerstones in Model-Driven Engineering (MDE). Therefore model analysis and quality improvement techniques like validation and verification of properties are crucial for the success of MDE.

We here employ the UML (Unified Modeling Language) and OCL (Object Constraint Language) for formulating models and focus on analysis techniques for model properties regarding behavior in the context of the tool USE (UML-based Specification Environment) [8]. In particular, USE allows the developer to automatically construct test cases in form of system states (object diagrams) for UML and OCL models. This opens the option to validate models against informal expectations and to verify essential properties like model consistency. Behavior is expressed in our approach by OCL operation contracts, i.e., pre- and postconditions. In our approach, behavioral models are transformed into so-called filmstrip models [9] that explicitly express behavior through operation call objects. Behavior in filmstrip models is formulated with multiple snapshot objects being part of a single system state. Our approach offers a high degree of test automation through the use of so-called classifying terms [10] that partition the test input space into relevant equivalence classes and allow to select equivalence class representatives. We have not yet studied an approach that combines filmstripping with classifying terms.

One main contribution of this paper is introducing the option to prove behavioral consistency of a UML and OCL model through the construction of a test case: the operations contracts (pre- and postconditions) together with the invariants are shown to be satisfiable. The notion behavioral consistency is understood here in the sense that there exists a sequence of operation calls, in which all operations occur, and in which all invariants and all pre- and postconditions are satisfied. To the best of our knowledge, showing consistency of operation contracts considered together with class invariants has not been studied so far.

The rest of this paper is organized as follows. Section 2 explains the basics of our approach to behavior modeling. Section 3 sketches behavior validation and verification options. Section 4 discusses the lessons learnt from our example case. The paper is closed with related work and a short conclusion.

2 Filmstripping UML and OCL Models

2.1 Application Model

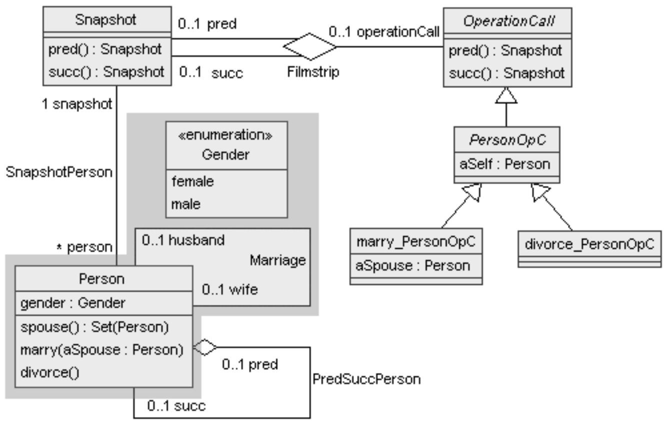
Our starting example model MarriageWorld is a toy model as displayed in the grey part of Fig. 1. We call this grey part the *application model* as we will later use a transformation of it, the so-called *filmstrip model.* This application model comprises invariants and operation contracts including operation frame conditions that express which model parts are left unchanged by a particular operation. These complete operation contracts may be seen as an alternative to an imperative operation implementation which is not needed in our approach.

The application model encompasses one set-valued OCL query operation and OCL invariants and operation contracts. We show the single invariant and the marry contract. The divorce contract is formulated analogously.

2.2 Filmstrip Model

The filmstrip model results from a transformation of the application model where classes and associations are added (the non-grey part in Fig. 1) and in particular pre- and postconditions are transformed into invariants. An application model sequence of operation calls and intermediate object diagrams correspond to a single object diagram in the filmstrip model. Operation calls in the application model are represented as operation call objects in the filmstrip model. For example, the top right sequence diagram in Fig. 2 corresponds to the top left object diagram. The transformation is realized through a USE plugin. The resulting filmstrip model is a plain UML class model with invariants only.

The filmstrip model organizes application model object diagram sequences into a linear sequence of so-called snapshots (the Snapshot objects in Fig. 2) encompassing the respective objects from the different application model object diagrams. The filmstrip model introduces further operations on class Person not shown in Fig. 1: succPlus(), succStar(), predPlus() and predStar() for the transitive closure and transitive-reflexive closure of the roles succ and pred, respectively. The role succ, for example, points from one object to its next reincarnation in the following snapshot. These operations employ the OCL operation closure and are essential to navigate forward and backward between different 'points in time from the application model point of view'. Generated invariants take care that the filmstrip model object diagrams behave correctly, e.g., that



```
spouse():Set(Person)=
  if wife->notEmpty and husband->notEmpty then Set{wife,husband} else
  if wife->notEmpty then Set{wife} else
  if husband->notEmpty then Set{husband} else Set{} endif endif endif
context Person inv traditionalRoles:
  (gender=#female implies wife->isEmpty) and
  (gender=#male implies husband->isEmpty)
context Person::marry(aSpouse:Person)
  pre unmarried:
    self.spouse()->isEmpty and aSpouse.spouse()->isEmpty and
    Set{self.gender,aSpouse.gender}=Set{#female,#male}
  post married:
    Set{aSpouse}=self.spouse() and Set{self}=aSpouse.spouse()
  post personUnchangedExceptSet: let x=self.spouse()->including(self) in
    Person.allInstances@pre=Person.allInstances and
    Person.allInstances->forAll(p|
      (p.gender@pre=p.gender) and
      (x->excludes(p) implies p.wife@pre=p.wife) and
      (x->excludes(p) implies p.husband@pre=p.husband))
```

Fig. 1. Example application model (grey) and filmstrip model.

an object and its reincarnations build a cycle-free pred-succ chain (for example, in the left top part of Fig. 2 the reincarnations of person4 are person12 and then person2).

3 Analysing Model Behavior

3.1 Configurations and Classifying Terms

The USE model validator [12] constructs object diagrams for a UML class diagram enriched by OCL invariants and is based on relational logic [11]. The

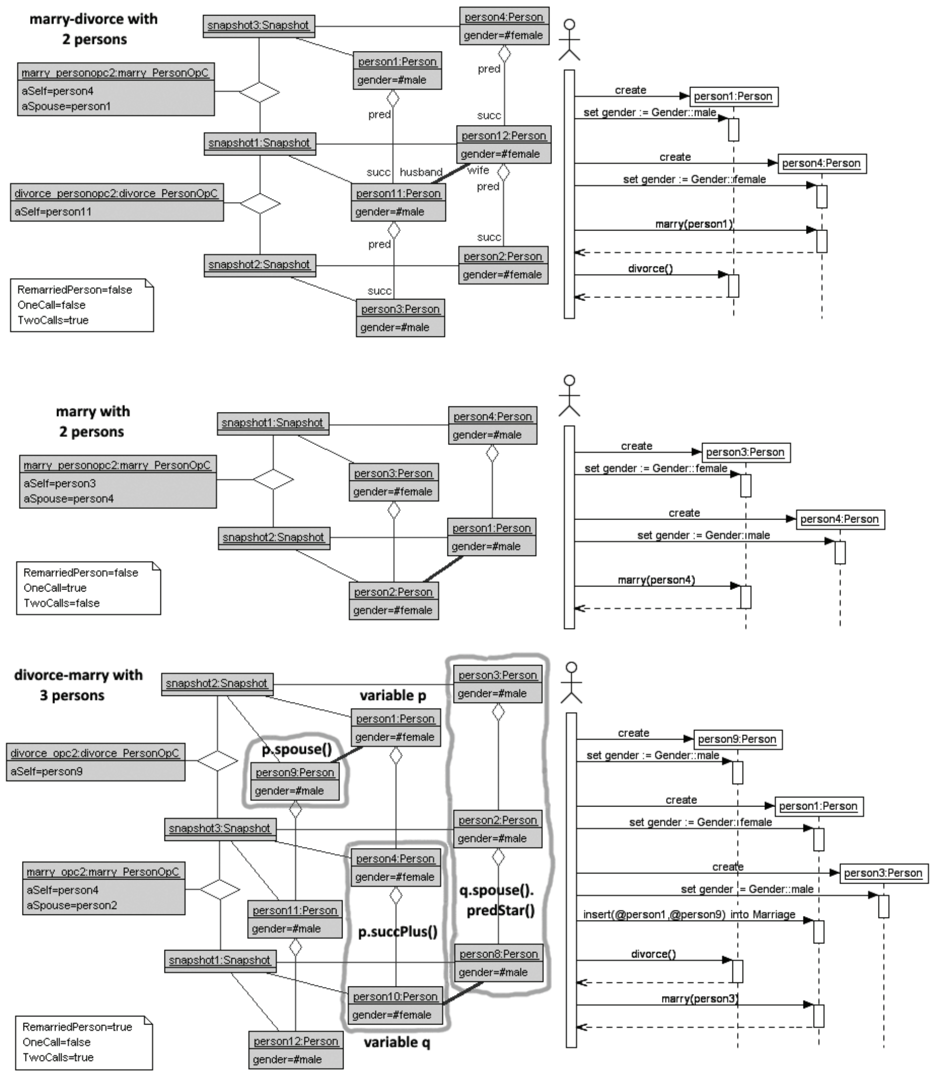
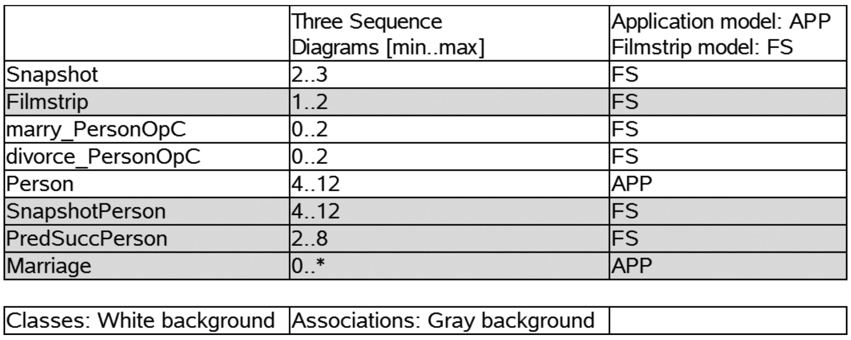


Fig. 2. Generated behavior scenarios.

validator has to be instructed by a so-called *configuration* that determines how the classes and associations are *populated*. For every class a mandatory upper bound and an optional lower bound for the number of objects is given. For every association optional lower and upper bounds can be stated.

Fig. 3 shows the used configuration (association bounds with grey background) in the example. The configuration determines scenarios having (a) two snapshots with one operation call in between or (b) three snapshots with two operation calls in between. In each snapshot two, three or four persons can be present. The kind of operation call (marry or divorce) is left open as both operation call classes are allowed to have 0, 1 or 2 elements.



	Three Sequence Diagrams [min..max]	Application model: APP Filmstrip model: FS
Snapshot	2..3	FS
Filmstrip	1..2	FS
marry_PersonOpC	0..2	FS
divorce_PersonOpC	0..2	FS
Person	4..12	APP
SnapshotPerson	4..12	FS
PredSuccPerson	2..8	FS
Marriage	0..*	APP

Classes: White background	Associations: Gray background	

Fig. 3. Used model validator configuration.

A central ingredient in our approach are so-called classifying terms that allow the developer to construct test cases for a UML and OCL model in a systematic way. They classify and determine test equivalence partitions. The test cases are specified by a set of OCL query expressions on the UML class model, the classifying terms, and are manifested in form of object diagrams. The constructed object diagrams will show *differences* with respect to at least one *classifying term*. The classifying terms used here are as follows.

```
[RemarriedPerson]
  Person.allInstances->exists(p | p.spouse()->size=1 and
    p.succPlus()->exists(q | q.spouse()->size=1 and
      q.spouse().predStar()->excludesAll(p.spouse())))
[OneCall]  OperationCall.allInstances->size=1
[TwoCalls] OperationCall.allInstances->size=2
```

The classifying terms refer to the filmstrip model. The three boolean typed classifying terms ask for scenarios where the terms either yield false or true and can be understood as follows: [RemarriedPerson] one person exists that is married differently in two different snapshots; the variable `p` fixes that person, and the variable `q` contains a later incarnation of `p`; the two snapshots are `p.snapshot` and `q.snapshot`; furthermore the previous incarnations of the spouses of `q` must be disjoint from the spouses of `p`; [OneCall] there is exactly one operation call; [TwoCalls] there are exactly two operation calls.

The classifying terms [OneCall] and [TwoCalls] cannot both be true at the same time. As the configuration allows one or two operation calls, the classifying terms [OneCall] and [TwoCalls] assert that at least one scenario with one operation call and at least one scenario with two operation calls will be constructed. As three boolean terms are given here, potentially $2^3 = 8$ scenarios will be constructed, but not all options can be realized.

3.2 Constructing Sequence Diagrams

In Fig. 2 the found three solutions for the configuration and the classifying terms are shown: in the left in form of the found filmstrip object diagram and in the right in form of a sequence diagram from the application model corresponding to the filmstrip object diagram. The links from the application model (the Marriage links) are displayed with fat lines. From the 8 possible combinations of the 3 boolean terms only 3 are feasible.

The found scenarios illustrate the behavior of the UML and OCL model with concrete behavior manifestations. Thus configurations and classifying terms support the developer in exploring model behavior. The construction of sequence diagrams with two operation calls employing different operations (marry and divorce) furthermore verifies the consistency of the invariants and the operation contracts taken together, because a scenario has been found where all invariants and all pre- and postconditions are valid.

The classifying term [RemarriedPerson] is true only in the third generated scenario in Fig. 2. In the figure, the distinguishing classifying term values are shown in the comment nodes. In addition, the objects that must be substituted for the OCL variables `p` and `q` are indicated. Furthermore, the values of the central OCL expressions from [RemarriedPerson] are marked: `p.succPlus()` and `q.spouse().predStar()` are disjoint collections.

Figure 4 gives an overview on the approach taken in this contribution. First the application model is filmstripped which results in the filmstrip model. Then the classifying terms and the model validator configuration (possessing parts for the application and for the filmstrip model) are used to generate object diagrams. The filmstrip parts of the configuration can guarantee, for example, that all (application model) operations calls occur exactly once. Note that all OCL quantifications (for example in the invariants or the classifying terms) range over finite domains, e.g., over classes or over constructed finite collections of datatype values. Thus a mapping into SAT solvers is feasible.

As already mentioned, in our approach verification of properties like consistency considering invariants and operation contracts together is possible. We are not aware of approaches that explore behavioral consistency focussing on the interplay of operation contracts and invariants.

4 Lessons Learnt

The small study shows that with the already existing options interesting results can be obtained, but the study also revaled topics for future work.

– The access to objects and their incarnations is rather involved through complicated OCL expressions. More standard filmstrip operations are needed to simplify such expressions. Currently, there is also only a rather involved access to the operations in which the objects occur. Again, more standard filmstrip operations can help to make the expressions easier.

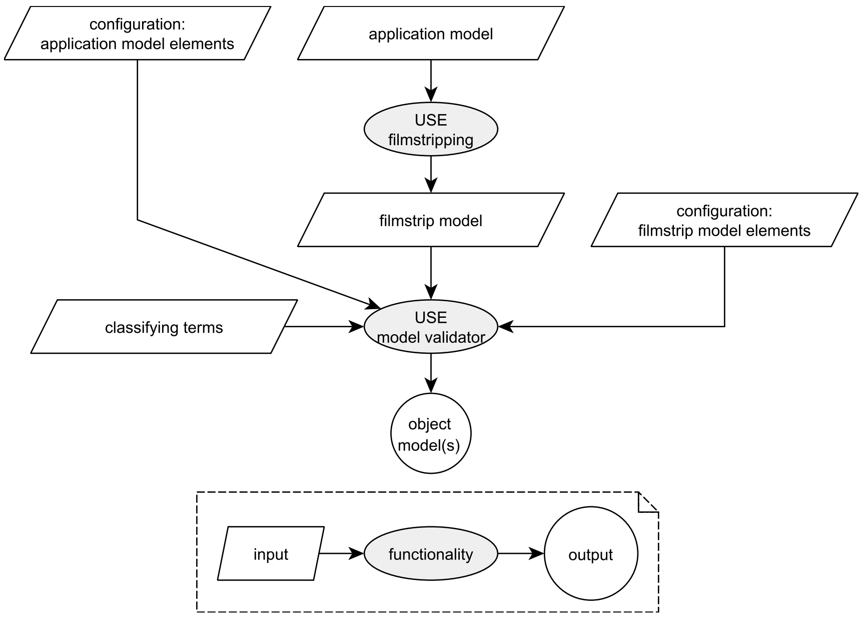
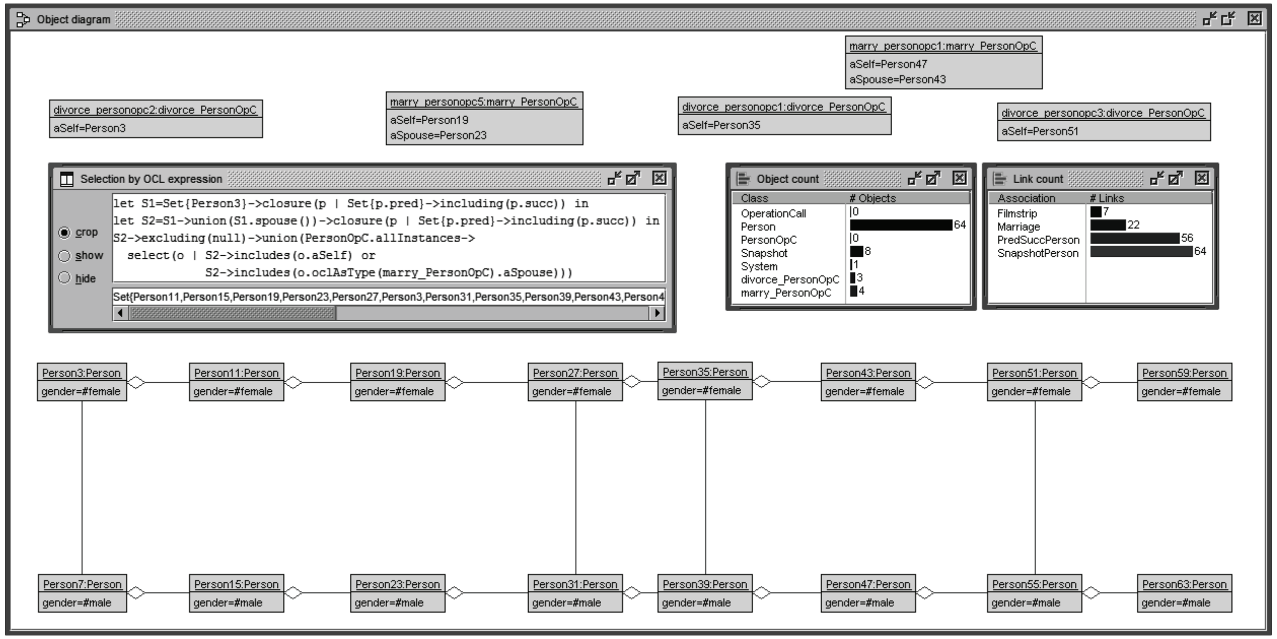


Fig. 4. Overview on combination of filmstripping with classifying terms.

- Filmstrip configurations should distinguish between application objects (e.g. from class Person) and filmstrip objects (from classes Snapshot or OpC). Dependencies between configuration items should be handled automatically (e.g. [Number of Snapshot objects]-1 = Number of OpC objects).
- An automatic layout for filmstrip object diagrams (in which OpC objects are placed between snapshots, and snapshots are placed above application model objects) would be helpful. The automatic representation of filmstrip object diagrams as application model sequence diagrams is needed.
- The approach scales to achieve larger scenarios. There is large potential for optimization in the filmstrip model through the construction of (Snapshot,ApplicationModelObject) templates that only need to be completed by the model validator. For example, if n operation calls on m application model objects are wanted, one can pre-compute the needed n+1 snapshot objects each connected to m application objects and one could establish the proper links automatically. Fig. 5 sketches an example with 7 operation calls and 64 Person objects. The scenario was obtained by creating the snapshots and the application objects explicitly with a script; then the model validator had only to find the proper operation call objects and the attribute values.

5 Related Work

Many approaches and tools have been proposed to support UML validation and verification. Using Constraint Logic Programming as the underlying formalism, *UMLtoCSP* [5] can automatically check several correctness properties of a UML



```
let S1=Set{Person3}->closure(p | Set{p.pred}->including(p.succ)) in
let S2=S1->union(S1.spouse())->closure(p | Set{p.pred}->including(p.succ)) in
S2->excluding(null)->union(PersonOpC.allInstances->
  select(o | S2->includes(o.aSelf) or
             S2->includes(o.oclAsType(marry_PersonOpC).aSpouse)))
```

```
Set{Person11,Person15,Person19,Person23,Person27,Person3,Person31,Person35,Person39,Person43,Person4
```

Class	# Objects
OperationCall	0
Person	64
PersonOpC	0
Snapshot	8
System	1
divorce_PersonOpC	3
marry_PersonOpC	4

Association	# Links
Filmstrip	7
Marriage	22
PredSuccPerson	56
SnapshotPerson	64

Fig. 5. Large example with 8 snapshots and 7 operation calls; irrelevant objects hidden with an OCL expression; ‘Object count’ and ‘Link count’ indicate the complete state.

class diagram enriched with OCL constraints. That approach also handles OCL contracts [4], however operation call and snapshot sequences are not treated. In [6], *DresdenOCL* is employed as a tool for OCL constraints verification. The

HOL-TestGen tool presented in [3] is able to automatically generate test cases based on a transformation of UML class models enhanced with OCL constraints into higher-order logic. In [16], an approach for the validation both static and dynamic aspects, e.g., class liveness property, of a UML model is introduced using a toolset based on Abstract State Machines. The approach in [15] describes a transformation of class diagrams and OCL constraints into first-order logic and methods for verifying properties such as class liveliness. As presented in [1], *UML2Alloy* can automatically transform UML model into Alloy, and then test models for consistency using the Alloy analyzer. Another approach is presented in [13], which uses a deep embedding strategy to allow for a transformation of more UML class diagram features. The approach in [2] studies model tests based on (positive and negative) UML sequence diagrams in connection with state charts. The work in [14] describes an approach to transform structural properties of class diagrams into Alloy and then verifies OCL constraints by finding valid snapshots of models. Details of the example model of this contribution, e.g., the complete filmstripped USE model, can be found in [7]. In contrast to the other mentioned approaches, our approach is the only one supporting systematic test case construction with classifying terms.

6 Conclusion and Future Work

We have presented an approach for automatically validating and verifying behavioral model features. In our approach it is possible to check behavioral model consistency and, in principle, behavioral model equivalence on the basis of generated test cases. The implementation of our model validator is based on the relational model finder Kodkod which in turn is translating into SAT solvers.

Future work will address a number of topics. Optimizing our translation and providing more SAT solvers could improve validation and verification efficiency. Further state space reduction techniques have to be considered. The current user interface in our tool for behavioral feature support is minimal. Advanced user feedback in the case that a verification task could not be successfully finished is desirable. Support for building frame conditions, automatic transformation into the filmstrip model as well as advice and proposals for configuration settings for the filmstrip model configuration are needed. Templates for verification tasks could minimize the developer interaction and inspire the developer for checking crucial properties. Expressing properties in an OCL version with temporal operators would increase readability and expressiveness of behavioral properties. Last but not least, larger case studies should give more feedback on the practicability of the approach.

References

1. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: On Challenges of Model Transformation from UML to Alloy. Software and System Modeling **9**(1) (2010) 69–86

2. Brosch, P., Egly, U., Gabmeyer, S., Kappel, G., Seidl, M., Tompits, H., Widl, M., Wimmer, M.: Towards Scenario-Based Testing of UML Diagrams. In Brucker, A.D., Julliand, J., eds.: Proc. 6th Int. Conf. TAP (2012). LNCS 7305 (2012) 149–155
3. Brucker, A., Krieger, M., Longuet, D., Wolff, B.: A Specification-Based Test Case Generation Method for UML/OCL. In Dingel, J., Solberg, A., eds.: Models in Software Engineering, Springer, LNCS 6627 (2010) 334–348
4. Cabot, J., Clarisó, R., Riera, D.: Verifying UML/OCL Operation Contracts. In Leuschel, M., Wehrheim, H., eds.: Proc. 7th Int. Conf. IFM 2009. LNCS 5423, Springer (2009) 40–55
5. Cabot, J., Clarisó, R., Riera, D.: On the Verification of UML/OCL Class Diagrams using Constraint Programming. Journal of Systems and Software **93** (2014) 1–23
6. Demuth, B., Wilke, C.: Model and Object Verification by Using Dresden OCL. In: Proc. Russian-German WS Innovation Information Technologies: Theory and Practice. (2009) 687–690
7. Gogolla, M., Hilken, F., Doan, K.H., Desai, N.: Addendum to Checking UML and OCL Model Behavior with Filmstripping and Classifying Terms. Technical report, University of Bremen (2017) http://www.db.informatik.uni-bremen.de/publications/intern/GHDD2017ADD.pdf.
8. 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
9. 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
10. Gogolla, M., Vallecillo, A., Burgueno, L., Hilken, F.: Employing Classifying Terms for Testing Model Transformations. In Cabot, J., Egyed, A., eds.: Proc. 18th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2015), ACM (2015) 312–321
11. Jackson, D.: Software Abstractions - Logic, Language, and Analysis. MIT Press (2006)
12. 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
13. Maoz, S., J.-O.Ringert, Rumpe, B.: CD2Alloy: Class Diagrams Analysis Using Alloy Revisited. In Whittle, J., Clark, T., Kühne, T., eds.: Model Driven Engineering Languages and Systems, MODELS 2011, Springer, LNCS 6981 (2011) 592–607
14. Massoni, T., Gheyi, R., Borba, P.: A UML Class Diagram Analyzer. In: In 3rd Int. Workshop Critical Systems Development with UML. (2004) 143–153
15. Queralt, A., Teniente, E.: Reasoning on UML Class Diagrams with OCL Constraints. In Embley, D.W., Olivé, A., Ram, S., eds.: Conceptual Modeling - ER 2006, Springer, LNCS 4215 (2006) 497–512
16. Shen, W., Compton, K., Huggins, J.: A Toolset for Supporting UML Static and Dynamic Model Checking. In: Proc. Computer Software and Applications Conference (COMPSAC 2002). (2002) 147–152