Executing Models by Filmstripping: Enhancing Validation by Filmstrip Templates and Transformation Alternatives*

Nisha Desai
Department of Computer Science
University of Bremen
D-28334 Bremen, Germany
nisha@informatik.uni-bremen.de

Martin Gogolla
Department of Computer Science
University of Bremen
D-28334 Bremen, Germany
gogolla@informatik.uni-bremen.de

Frank Hilken
Department of Computer Science
University of Bremen
D-28334 Bremen, Germany
fhilken@informatik.uni-bremen.de

***Abstract*—This paper discusses an approach to execute models with filmstrip templates in order to enhance the validation and execution process so that model execution time is decreased. A filmstrip template identifies recurring model parts. When such recurring model parts are constructed only once, model validation time is reduced. We also improve our previous filmstrip approach by employing more efficient architectures for the needed filmstrip transformation. By performing a feasibility study we check and show the performance of our filmstrip templates for different architectures and compare execution times for particular test cases. Among the developed and tested seven architectures, three show particular good results with respect to execution and validation time.**

I. Introduction

Recently, modeling languages as UML and OCL and executable models [1], [2] are gaining more and more importance in designing systems. For a given application model, it is crucial to validate and verify system properties during the design phase. A variety of validation and verification techniques are currently available. For the purpose of model behavior validation, the tool USE [3] can be employed to transform a given UML and OCL application model into an equivalent so-called filmstrip model [4]. In USE, a model validator can automatically generate a valid object diagram by using configurations, and thus automatic test case generation is possible. As systems and their models are getting more complex, test scenarios are getting larger, and therefore the execution time for model validation becomes crucial.

In this paper, we introduce filmstrip templates for our filmstrip approach in order to enhance the validation and execution process. A filmstrip template reduces the work of our model validator by identifying recurring model parts. Thereby validation time is reduced. Another improvement of our previous filmstrip approach is to employ more efficient architectures for the needed filmstrip transformation. Currently, the transformation of a filmstrip model is realized using a ternary association. But, there are other possible architectures for the filmstrip realization using binary association, aggregation, or composition. By performing a feasibility study we check and show the performance of the filmstrip templates for different architectures and compare the execution time of particular test cases.

The rest of this paper is organized as follows. Section II provides some background on filmstripping. Section III describes different architectures for filmstripping. Section IV introduces and explains the approach enhancing the validation process, i.e., the concepts behind filmstrip templates. Section V and VI illustrate the execution and the results of the comparative study that we have conducted. Related work is addressed in Sect. VII. Finally, in Sect. VIII, the paper is closed with conclusions and some future work.

II. Filmstripping

An ordinary UML and OCL application model can involve structural aspects in form of OCL invariants and dynamic aspect in form of operation pre- and postconditions. The model validator in the tool USE is designed for structural analysis of UML class diagrams. In order to validate dynamic aspects of the model, our filmstrip approach is used. Filmstripping transforms a given UML and OCL model which is comprised of invariants and pre- and postconditions into an equivalent model which possesses only invariants. The transformed model is called a filmstrip model. This transformed model involves only structural aspects and can be validated with the USE model validator. Roughly speaking, system dynamics is expressed in the filmstrip model by different explicit objects representing operation calls. A sequence of operation calls and object diagrams of an application model corresponds to a single object diagram of the filmstrip model [4].

The filmstripping approach can be explained best in terms of an example. Figure 1 shows essential parts of a *library model* [5] in which users can borrow and return book copies. The original application model is indicated in a gray-shaded style, namely the classes `User` and `Copy` together with the association `Borrows` in the class diagram, and the small sequence diagram represents the application model. Structural

*Research paper.

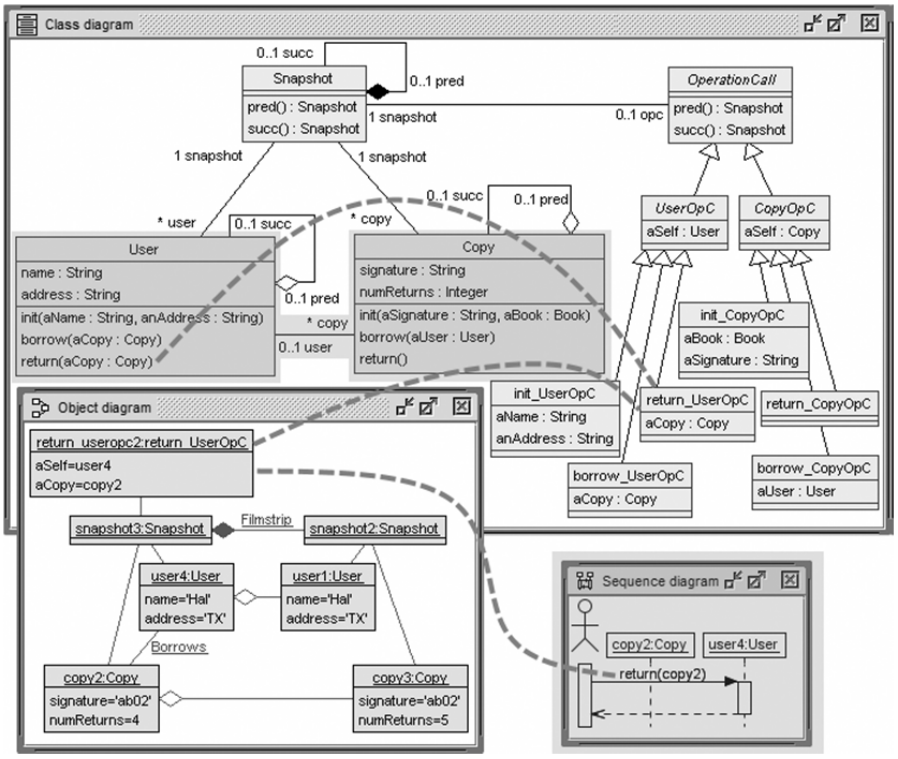

Fig. 1. Application model and filmstrip model (Architecture D in Fig. 2).

and dynamic aspects of this application model are completely described with OCL invariants and pre- and postconditions.

The application model is automatically transformed into the filmstrip model: the non-gray shaded classes and the object diagram in Fig. 1. In essence, the application model sequence diagram becomes a filmstrip model object diagram. `Snapshot` objects explicitly allow to capture single system states from the application model. Operation call objects (suffix OpC) describe operation calls from the application model. Basically, each operation is transformed into an `OperationCall` class with attributes for the `Self` objects and for the operation parameters. Thus, for example, the call `user4.return(copy2)` from the sequence diagram is represented by the object `return_useropc2` in the filmstrip object diagram. The effect of the operation call is represented by the differences between the left and the right snapshot: The `return` operation call removes the `Borrows` link and increases the attribute `numReturns`. The two `User` and the two `Copy` objects represent different object states before and after the operation call. One could say that the object `copy3` is a later incarnation of the object `copy2`. The *library model* contains another class `Book` not shown in Fig. 1.

III. Architectures for Filmstripping

The transformation into a filmstrip model introduces new classes for snapshots and for operation calls. Furthermore, proper directed connections between these snapshots and operation calls are realized through links. Snapshots and operation calls play a central role in the filmstripping approach, because they provide the glue that keeps the items together. As mentioned earlier, currently a ternary association is used to link snapshots with each other and with related operation calls. However, there are other possible architectures through which snapshots and operation calls can be linked. Figure 2 shows seven different possible architectures.

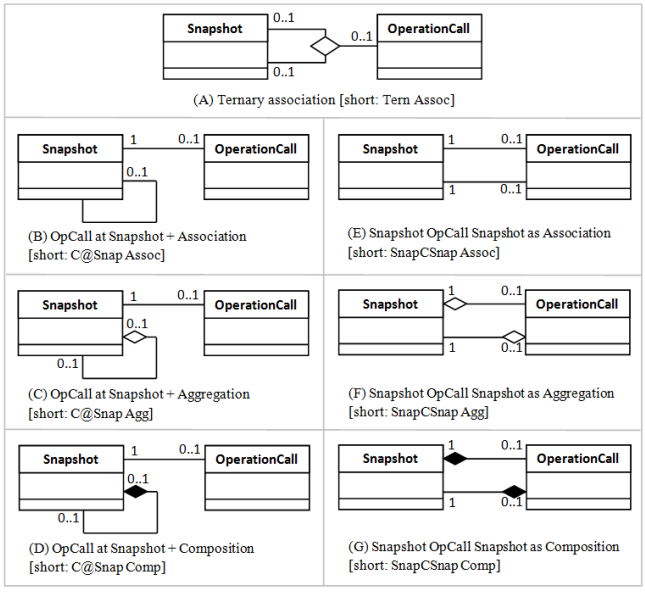

Fig. 2. Different filmstrip architectures for the transformation.

Figure 2(A) shows the use of a ternary association

to connect Snapshot to Snapshot as well as to OperationCall. Figure 2(B) displays a possibility to connect Snapshot to Snapshot and to an OperationCall by means of two associations. Here the first association connects the Snapshot sequence and the other connects the OperationCall to Snapshot which represents the pre-state of the OperationCall. Figures 2(C) and (D) are almost same as (B), except that the first uses aggregation and the second uses composition instead of an association to connect Snapshot to Snapshot. Figure 2(E) shows another approach to connect the classes using two associations. This time the Snapshot class is connected to the OperationCall and in return, this OperationCall class is again connected to the Snapshot, which represents the post-state of the OperationCall. Figures 2(F) and (G) are more or less similar to (E), except that the first uses aggregation and the second uses composition instead of an association.

All newly introduced architectures (B)-(G) can be used to realize filmstripping and are interchangeable, but they all have some pros and cons. The architectures (B), (C), and (D) have a link which connects Snapshot to Snapshot directly which makes the navigation from Snapshot to Snapshot independent from the OperationCall class. But it could be possible that a post-state is generated without an OperationCall. Thus OCL invariants are used to overcome this situation and to execute the architectures successfully. In the case of architectures (E), (F) and (G), the Snapshot class is connected via the OperationCall class. With these architectures, it is not possible to navigate between the Snapshot objects directly, and this requires OCL expressions to navigate from Snapshot to Snapshot object. The use of aggregation and composition links in the architectures ensure that the connections are cycle-free.

IV. Enhancing filmstripping with templates

Within the filmstrip approach, the USE model validator [6] uses configurations (which describe a number of application and filmstrip objects and links) and constructs an object diagram for the filmstrip model. If the number of snapshots, operation calls and application objects are known, one can introduce filmstrip templates. Figure 3 gives an overview on the validation process using filmstripping and the template approach.

The template consists of snapshot objects and application objects without attribute values, but with already established connecting links between them. For example, if n operation calls on m application model objects are known from the configuration, one can pre-compute the needed $n+1$ snapshot objects each one being connected to m application objects and one can establish the needed links automatically. Figure 4 sketches an example with 2 operation calls and 9 application objects. When the developer has made the choice for the number of snapshots and the number of objects in each application class, the filmstrip template approach constructs a partial object diagram that has to be only completed by the USE model validator. So the model validator has only a few

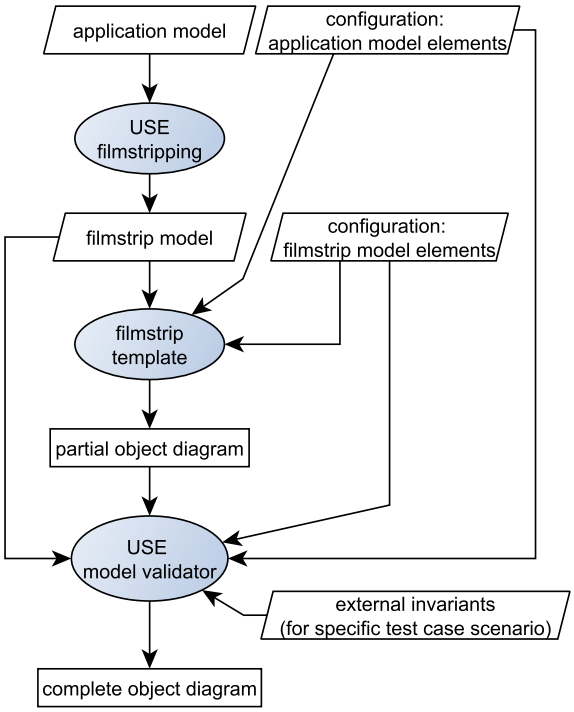

Fig. 3. Overview on validation process with filmstrip templates.

elements to construct in order to achieve a complete object diagram. This approach enhances the validation process and leads to a faster execution time.

Filmstrip object diagrams using any of the architectures have snapshot and application objects. These objects (without attribute values) and their links (object-object and object-snapshot) can be defined in the filmstrip template. Apart from this, in the case of architectures (B), (C), and (D), the Snapshot connects with itself without a need for an OperationCall. This allows to define snapshot-snapshot links in the filmstrip template for the architectures (B), (C) and (D).

Figure 4 shows a template for a filmstrip object diagram which is relying on architecture (B). The model elements shown in black (snapshot and application model objects without attribute values and links) represent the filmstrip template and the part highlighted in gray has to be generated by the model validator. So the model validator has only to find the proper OperationCall objects, application model links, and attribute values. The overhead for handling the snapshot and application model objects are effectively reduced.

V. Study execution

In order to check the performance of the filmstrip template approach and the different filmstrip architectures in terms of execution time, a feasibility study has been performed. The test cases have been executed with and without filmstrip templates for the filmstrip architectures. The filmstrip transformation using ternary filmstrip architecture (A) is already available in

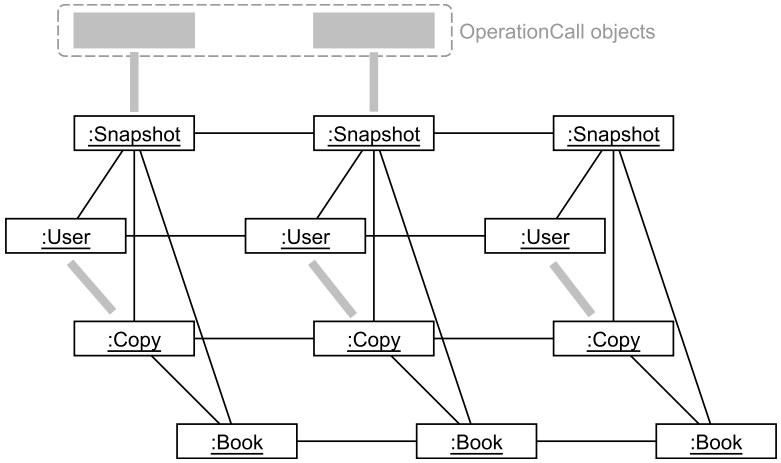

Fig. 4. Example template in the filmstrip approach (Architecture (B) from Fig. 2).

the tool USE. The other shown architectures (B)-(G) are now implemented and can be accessed directly. Here two test cases, one for the *library model* and one for the *concurrentAppend* model [7] have been considered for the study.

The test cases used are shortly described in Table I. Each test case constructs a scenario based on initial and final snapshot conditions expressed as OCL invariants. The configurations mentioned in the table restrict the number of allowed objects in the respective class and therefore restrict the operation calls in a corresponding application model sequence diagram. Based on the stated configurations and invariants, the USE model validator constructs a valid object diagram [8]. The filmstrip templates for this study have been constructed manually from the configurations for all architectures. The execution time for generating the object diagrams with and without filmstrip templates are measured. In order to obtain accurate execution times, each test case has been run five times for each of the seven filmstrip architectures (A)-(G). We have calculated a trimmed mean by dropping the highest and lowest values from the collected execution times [9]. A detailed description of the test cases is given below:

A. Test case 1: library model

The scenario of test case 1 is "Initially a copy of a book is in the library and the copy has a number of returns equal to zero, and in the final state, the copy should have a number of returns equal to three".

```
context Copy inv numReturnZero:
Snapshot.allInstances()->
  any(s|s.pred()=null).copy->
    forAll(c|c.numReturns=0)

context Snapshot inv noLinkUserCopy:
Snapshot.allInstances()->
  any(s|s.pred()=null).user.copy->
    size()=0

context Copy inv numReturnThree:
Snapshot.allInstances()->
  any(s|s.succ()=null).copy->
    forAll(c|c.numReturns=3)
```

There are three invariants specified in this test case for generating the scenario. The first invariant `numReturnZero` and the second invariant `noLinkUserCopy` are for the initial snapshot, and the third one is for the final snapshot: First the number of returns of the copy is required to be zero; second, the copies in the library are restricted; the third invariant `numReturnThree` concerns the number of returns of the copy.

The operation specification for this test case has been given in the range 0..6. Based on the externally specified invariants, the model validator chooses the operation calls. The configuration is mentioned below:

```
Snapshot = 7..7
borrow_UserOpC = 0..6
return_CopyOpC = 0..6
borrow_CopyOpC = 0..6
return_UserOpC = 0..6
```

B. Test case 2: concurrentAppend model

The scenario chosen for test case 2 is "Initially three cells exist with the values in a list and one new cell (value: 5) is appended to the list and in the final state, there should not be an unfinished append".

As test case 1, there are also external invariants that are used for generating the scenario and based on the specified invariants the model validator chooses the operation calls. In this test case, the operation specification has been given in the range 0..4. The configuration is mentioned below:

```
Snapshot = 5..5
append_AppendOpC = 0..4
found_AppendOpC = 0..4
next_AppendOpC = 0..4
return_AppendOpC = 0..4
```

TABLE I
TEST CASES USED FOR THE FEASIBILITY STUDY.

	Test case 1 - library model	Test case 2 - concurrentAppend model
Application configuration	1 User, 1 Book, 1 Copy	3 Cells, 1 Append
Invariants	Initial condition: Copy is in library and number of returns is zero. Final condition: Number of returns of the copy is three.	Initial condition: Mention three Cells and one Append with values. Final condition: Append should be finished.
Filmstrip configuration	Snapshot = 7..7 borrow_UserOpC = 0..6 return_UserOpC = 0..6 borrow_CopyOpC = 0..6 return_CopyOpC = 0..6	Snapshot = 5.. 5 append_AppendOpC = 0..4 return_AppendOpC = 0..4 found_AppendOpC = 0..4 next_AppendOpC = 0..4
Expected Result	3 { {User|Copy}_borrow and {User|Copy}_return} operation calls.	Next-Next-Append-Return operation calls.

TABLE II
RESULT COMPARISION - TEST CASE 1.

Archi-tecture	A Tern Assoc	B C@Snap Assoc	C C@Snap Agg	D C@Snap Comp	E SnapCSnap Assoc	F SnapCSnap Agg	G SnapCSnap Comp
Test case 1	32.73 min	0.43 min	1.34 min	1.41 min	2.50 min	3.23 min	3.28 min
	0.13 min	0.07 min	0.09 min	0.08 min	0.15 min	0.16 min	0.16 min

TABLE III
RESULT COMPARISION - TEST CASE 2.

Archi-tecture	A Tern Assoc	B C@Snap Assoc	C C@Snap Agg	D C@Snap Comp	E SnapCSnap Assoc	F SnapCSnap Agg	G SnapCSnap Comp
Test case 2	90.47 min	29.13 min	20.00 min	20.46 min	20.15 min	29.24 min	27.90 min
	0.33 min	0.21 min	0.22 min	0.23 min	0.25 min	0.29 min	0.29 min

TABLE IV
AVERAGE EXECUTION TIME OF TEST CASES USING FILMSTRIP TEMPLATES.

Archi-tecture	A Tern Assoc	B C@Snap Assoc	C C@Snap Agg	D C@Snap Comp	E SnapCSnap Assoc	F SnapCSnap Agg	G SnapCSnap Comp
Test case 1	0.13 min	0.07 min	0.09 min	0.08 min	0.15 min	0.16 min	0.16 min
Test case 2	0.33 min	0.21 min	0.22 min	0.23 min	0.25 min	0.29 min	0.29 min
Sum	0.46 min	0.28 min	0.31 min	0.31 min	0.40 min	0.45 min	0.45 min

VI. STUDY RESULTS AND COMPARISON

The models for the efficiency check with respect to execution time have been studied using a laptop with an Intel Core i5-4210U processor running at 2.4GHz, 8GB of RAM and Java 1.8.0. The models were validated employing USE 4.2.0.

In this section, the results of the execution time for the test cases are discussed. The execution is performed with filmstrip templates and also without filmstrip templates for all architectures. Tables II and III show the final averaged execution times of each architecture for both the test cases and also the enhanced results (execution with filmstrip template) which are highlighted in a white-on-darkgray style. The results are compared with the original results (execution without filmstrip template). It can be concluded from the results that the enhancement approach proposed in this paper makes the execution of the validation process faster. In comparison to the time saving, the time needed to create a template manually is neglectable. Automatically creating these templates is part of future work. The reason for the better performance is that the overhead for handling the snapshot and application objects of the model validator is significantly reduced, as the filmstrip templates are constructed before the validation. This makes the execution and validation process faster.

Table IV shows the execution times of both the test cases with filmstrip templates and the comparison between the architectures. One can notice that architectures (B), (C), and (D) (highlighted in a black-on-lightgray style) are the fastest in both the test cases. The reason for this is that the snapshot-snapshot link do not depend on `OperationCall` objects, as was mentioned in the discussion of filmstrip templates for architectures (B), (C) and (D). This means that filmstrip templates of that architectures contain more elements compared to templates of other architectures. So the number of elements to be generated by model validator is again reduced, and that makes the execution smaller.

VII. RELATED WORK

Many approaches and tools have been proposed for UML and OCL model execution and model validation. In [10], the authors perform model execution by compiling the model to program code. They have proposed a model simulator tool

which translates the UML model to Java code and execution is performed on this generated code. In [11], the tool Matilda is introduced, which accepts a UML model as an input data and validates it against the UML metamodel. The tool generates a Java abstract syntax tree (JAST) from the input UML model and executes Java bytecode generated from JAST. In [12], the authors discuss about a code generation tool, which transforms UML models (classes and state machines) into C# source code and supports execution of the application corresponding to both structural and behavioral models. The above mentioned approaches are based on the generation of code from the UML models and execution is performed on this generated code. In contrast, our approach uses model transformation concepts and the execution is performed on this transformed model, i.e., the execution is performed at the modeling level. In [13], the authors present the tool Populo, which is used for executing and debugging UML class diagrams whose behavior is specified by the UML action language and described by activity diagram containing actions. In contrast, the behavior of the UML model is described by sequence diagrams using OCL invariants in our approach. In [14], a model transformation approach for validating UML models comprised of class and sequence diagrams is presented. A UML model is transformed into Testable Aggregate Model (TAM) for test case execution. In [15], a UML class diagram along with OCL constraints is transformed to Alloy code, and using the Alloy Analyzer, execution and validation is performed. In [16], the authors present an approach for the verification of EMF models annotated with OCL constraints. The input UML model along with the properties to be validated is transformed into a constraint satisfaction problem (CSP), and using a constraint solver, the CSP is executed and validated. In contrast to [14], [16] and [15], we transform a UML and OCL application model into a filmstrip model and we are using the model validator. Our approach provides a seemless integration of structure and behavior and is based on the support of full OCL.

VIII. Conclusion and future work

In this paper, the automatic transformation of an application model with invariants and pre- and postconditions into a filmstrip model in which the system behavior is represented with only invariants has been described. We have presented an approach for enhancing the validation process by introducing filmstrip templates and by proposing different architectures for the filmstrip transformation. Through experiments, we showed that the validation process using filmstrip templates is more efficient in terms of execution time compared to validation without templates. It can be concluded from our study that architectures (B), (C) and (D) yield better results and take less time than the other architectures.

Future work will study different and larger models and test cases with the same methodology as applied here in order to verify the current conclusions. We will also concentrate on the automatic generation of the template from developer specifications and implement developer support for this. Last but not least, means for distinguishing application and filmstrip elements in the configuration have to be developed.

References

[1] T. Mayerhofer, P. Langer, E. Seidewitz, and J. Gray, eds., *Proc. 1st Int. Workshop Executable Modeling co-located with (MODELS 2015)*, vol. 1560 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016.

[2] T. Mayerhofer, P. Langer, E. Seidewitz, and J. Gray, eds., *Proc. 2nd Int. Workshop Executable Modeling co-located (MODELS 2016)*, vol. 1760 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016.

[3] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-Based Specification Environment for Validating UML and OCL," *Science of Computer Programming*, vol. 69, pp. 27–34, 2007.

[4] M. Gogolla, L. Hamann, F. Hilken, M. Kuhlmann, and R. B. France, "From Application Models to Filmstrip Models: An Approach to Automatic Validation of Model Dynamics," in *Proc. Modellierung (MODELLIERUNG'2014)* (H. Fill, D. Karagiannis, and U. Reimer, eds.), pp. 273–288, GI, LNI 225, 2014.

[5] M. Gogolla, "Teaching Touchy Transformations," in *MODELS Educators' Symposium (EDUSYMP'2008)* (M. Smialek, ed.), pp. 13–25, Warsaw University, ISBN 83-916444-8-0, 2008.

[6] M. Kuhlmann and M. Gogolla, "From UML and OCL to Relational Logic and Back," in *Proc. 15th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2012)* (R. France, J. Kazmeier, R. Breu, and C. Atkinson, eds.), pp. 415–431, Springer, Berlin, LNCS 7590, 2012.

[7] A. Rensink, Á. Schmidt, and D. Varró, "Model Checking Graph Transformations: A Comparison of Two Approaches," in *Proc. Graph Transformations, 2nd Int. Conf., ICGT 2004* (H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, eds.), pp. 226–241, 2004.

[8] M. Gogolla and F. Hilken, "Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool," in *Proc. Modellierung (MODELLIERUNG'2016)* (A. Oberweis and R. Reussner, eds.), pp. 203–218, GI, LNI 254, 2016.

[9] T. Hu and S. Y. Sung, "A Trimmed Mean Approach to Finding Spatial Outliers," *Intell. Data Anal.*, vol. 8, no. 1, pp. 79–95, 2004.

[10] G. Dévai, M. Karácsony, B. Németh, R. Kitlei, and T. Kozsik, "UML Model Execution via Code Generation," in *Proc. 1st Int. Workshop Executable Modeling co-located with MODELS 2015* (T. Mayerhofer, P. Langer, E. Seidewitz, and J. Gray, eds.), pp. 9–15, 2015.

[11] H. Wada, J. Suzuki, M. M. B. Eadara, A. Malinowski, and K. Oba, "Design and Implementation of the Matilda Distributed UML Virtual Machine," in *Proc. 10th Int. Conf. Software Engineering Applications (SEA), 2006* (A. Cheng, ed.), 2006.

[12] A. Derezinska and R. Pilitowski, "Realization of UML Class and State Machine Models in the C# Code Generation and Execution Framework," *Informatica (Slovenia)*, vol. 33, no. 4, pp. 431–440, 2009.

[13] L. Fuentes, J. Manrique, and P. Sánchez, "Execution and Simulation of (Profiled) UML Models using Pópulo," in *Int. Workshop Modeling in Software Engineering, MiSE 2008* (J. M. Atlee, R. B. France, G. Georg, A. Moreira, B. Rumpe, S. Völkel, and S. Zschaler, eds.), pp. 75–81, 2008.

[14] O. Pilskalns, A. A. Andrews, A. Knight, S. Ghosh, and R. B. France, "Testing UML designs," *Information & Software Technology*, vol. 49, no. 8, pp. 892–912, 2007.

[15] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A Challenging Model Transformation," in *Proc. Model Driven Engineering Languages and Systems, 10th Int. Conf., MoDELS 2007* (G. Engels, B. Opdyke, D. C. Schmidt, and F. Weil, eds.), pp. 436–450, 2007.

[16] C. A. González, F. Büttner, R. Clarisó, and J. Cabot, "EMFtoCSP: A Tool for the Lightweight Verification of EMF Models," in *Proc. First Int. Workshop on Formal Methods in Software Engineering, FormSERA, 2012* (S. Gnesi, S. Gruner, N. Plat, and B. Rumpe, eds.), pp. 44–50, 2012.