Towards a Generic Verification Methodology for System Models

Robert Wille[1] Martin Gogolla[2] Mathias Soeken[1,3] Mirco Kuhlmann[2] Rolf Drechsler[1,3]
[1]Group for Computer Architecture, University of Bremen, 28359 Bremen, Germany
[2]Database Systems Group, University of Bremen, 28359 Bremen, Germany
[3]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{rwille,gogolla,msoeken,mk,drechsle}@informatik.uni-bremen.de

***Abstract*—The use of modeling languages such as UML or SysML enables to formally specify and verify the behavior of digital systems already in the absence of a specific implementation. However, for each modeling method and verification task usually a separate verification solution has to be applied today.**

In this paper, a methodology is envisioned that aims at stopping this "inflation" of different verification approaches and instead employs a generic methodology. For this purpose, a given specification as well as the verification shall be transformed into a basic model which itself is specified by means of a generic modeling language. Then, a range of automatic reasoning engines shall uniformly be applied to perform the actual verification. A feasibility study demonstrates the applicability of the envisioned approach.

I. Introduction & Background

Modeling languages such as the *Unified Modeling Language* (UML) [21] or the *Systems Modeling Language* (SysML) [22] together with textual constraints e.g. provided by the *Object Constraint Language* (OCL) [23] have been established to specify the design of complex systems. They provide different concepts such as class diagrams, sequence diagrams, or activity diagrams which are expressive enough to formally specify a complex system, but hide specific implementation details.

Since modeling languages permit formal descriptions, they additionally enable the verification of the respective specification already in the absence of a specific implementation. As a result, verification questions such as *"Does the conjunction of all constraints still allow the instantiation of a legal system state?"* or *"Is it possible to reach certain bad states, good states, or deadlocks?"* can be addressed already in the early design steps. These verification tasks are typically categorized in terms such as consistency, reachability, or independence [9].

For this purpose, many verification methods have been presented in the past. For example, theorem provers such as PVS [24], HOL-OCL/Isabelle [25], or KeY [26] have been applied. They perform a deductive derivation of the respective verification goal and have been shown to be quite powerful. However, they always require significant manual interaction as well as special knowledge and are therefore time- and cost-intensive.

Consequently, push button methods are desired. Methods based on automatic reasoning engines such as CSP solvers [3], description logic [27], Alloy [11], or SAT solvers [1], [6] have been shown to be quite promising. Here, the model together with the verification task are transformed into a valid input of a reasoning engine and are afterwards automatically solved by it.

978-3-9815370-0-0/DATE13/©2013 EDAA

However, the developments in the previous years led to an "inflation" of different verification approaches for designs given in terms of modeling languages. Often each approach addresses only a very dedicated verification task. Fig. 1(a) provides an (incomplete) overview. While e.g. [1] allows for consistency checking of class diagrams, this approach does not support sequence diagrams. That is, for each modeling method and each verification task usually a different verification solution has to be applied.

Moreover, complex systems are usually not specified by means of single diagrams only. In fact, a variety of diagrams of different types interact with each other. For example, while class diagrams specify the structure of a system, the behavior is defined by sequence diagrams. In order to verify such a specification, all diagrams need to be considered as a whole.

Finally, existing approaches are fixed to a certain reasoning engine. For example, the approach presented in [7] exploits CSP solvers, whereas e.g. in [6] SAT solvers find application. This is disadvantageous as reasoning engines may behave differently effective for various models. If additionally, new and better reasoning engines emerge in the future, existing transformations to the respective solver input have to be re-developed.

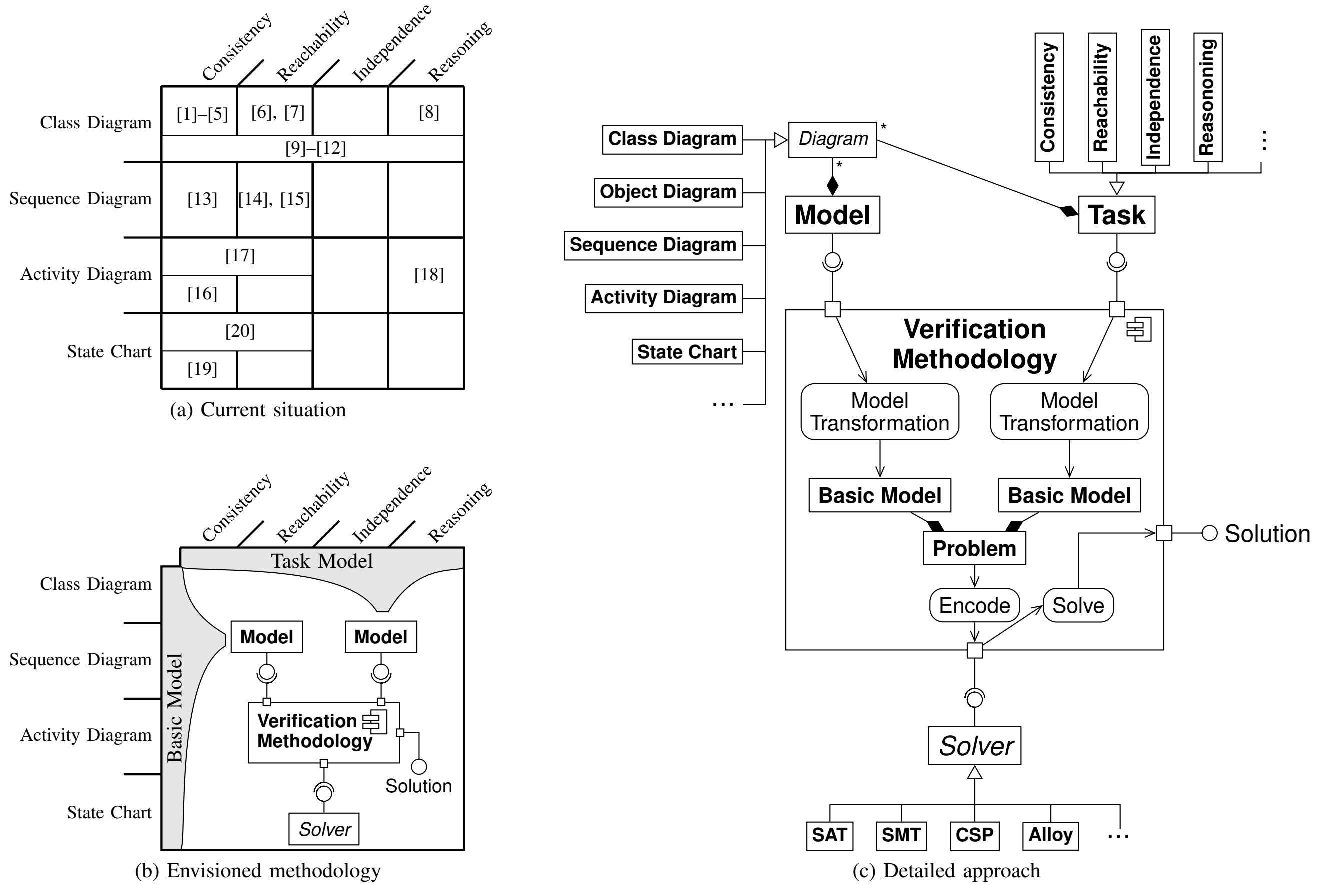
II. Proposed Idea

In this paper, we envision an approach that employs a generic methodology to verify specifications given in modeling languages. The general idea is illustrated in Fig. 1(b).

Instead of treating single diagram types or small combinations of them separately (as it has been done in the past; see Fig. 1(a)), we propose to transform them to a *basic model*. The basic model itself is specified by an atomic subset of UML and OCL constraints which is expressive enough to describe all constructs from languages such as UML, but small enough to allow for a flexible further processing.

Besides that, the verification tasks are modeled in the same language as the specification. For example, the question whether a certain bad state can be reached in a given model is formulated as a sequence diagram where the initial state and the considered bad state are provided, but the respective operation calls are left blank. These descriptions are also transformed into a basic model leading to a holistic verification environment.

The combination of the basic model and the task leads to the actual verification problem to be solved. Since both, the basic model and the task, are composed of an atomic number of UML constructs, transformations to the desired inputs for reasoning engines can be provided at moderate costs.



(a) Current situation

(b) Envisioned methodology

(c) Detailed approach

Fig. 1. Current verification approaches and envisioned generic methodology

Fig. 1(c) shows the general architecture of the envisioned methodology. In order to support a new diagram type, it is sufficient to provide a model transformation to the basic model without thinking of a specific verification task. Analogously, a new verification task is added by providing an appropriate model transformation to the basic model as well. Finally, a desired reasoning engine is incorporated by transforming the basic model into the respective language of the engine; again without thinking of precise diagram types to support.

III. Transformation to the Basic Model

Transforming arbitrary constructs of modeling languages into a basic model is obviously the biggest challenge of the envisioned methodology. In this section, we introduce the characteristics of the proposed basic model and illustrate by means of established UML descriptions how the required model transformations can be conducted.

A. Basic Model

The envisioned methodology aims at supporting a wide range of constructs from various modeling languages. At the same time, the transformation effort to the different reasoning engines should be as small as possible. Hence, the basic model needs to satisfy both of the following characteristics:

- *Universality*, i.e. for each construct an equivalent formulation in the basic model must exist.
- *Atomicity*, i.e. the constructs of the basic model should be limited to fundamental modeling concepts such that a uniform further processing as well as the flexibility of the overall framework is ensured.

We propose to realize these characteristics by restricting the basic model to a UML/OCL subset composed of (1) reduced UML class diagrams (e.g. allowing no inheritance, supporting only binary associations, etc.), (2) UML object diagrams following the same restrictions as their class diagram counterparts, and (3) textual constraints provided in OCL. These constructs are atomic enough to enable a flexible encoding for different reasoning engines. In fact, only the basic concepts of class diagrams and object diagrams need to be encoded for the addressed reasoning engine. OCL constraints can often be directly mapped to the required syntax. Examples of such encodings are e.g. available in [6], [7]. At the same time, this basic model is expressive enough to describe enhanced constructs from languages such as UML. This is illustrated in more detail in the following sections.

B. Transformation of Static Aspects

In order to transform enhanced UML constructs to the basic model, existing approaches should be exploited. For example, multiplicities of associations, aggregations, and compositions can be transformed as introduced in [28]. Similarly, arbitrary associations or classes of associations can be represented by binary associations together with OCL invariants. Inheritance

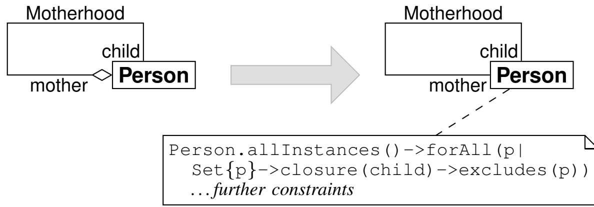
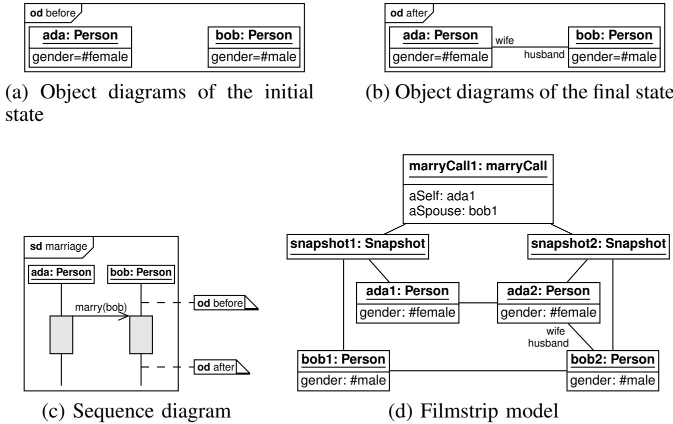


Fig. 2. Transformation of static aspects



(a) Object diagrams of the initial state

(b) Object diagrams of the final state

(c) Sequence diagram

(d) Filmstrip model

Fig. 3. Transformation of dynamic aspects

between two classes can be realized through delegation. Incomplete/complete- and overlapping/disjoint-constraints can be described through OCL invariants. As a result, many of the static aspects of the UML are already covered.

Example 1: Fig. 2 exemplary shows the transformation of an established UML construct (in this case, a reflexive aggregation) into the basic model.

C. Transformation of Dynamic Aspects

Although the basic model is restricted to static constructs, also UML constructs specifying dynamic behavior of a system should be transformed to it accordingly. To this end, the concept of the *filmstrip model* as introduced in [29] can be applied. Here, models including dynamic aspects are equivalently described by using only static constructs. Different states of the system are represented by means of so called *snapshot classes*. Operations which transform one system state to another one are specified through *method classes* and associations to the snapshot classes. Having that, the respective pre- and post-conditions of an operation are represented by OCL invariants of the method classes. The following example clarifies the idea.

Example 2: Fig. 3 shows a dynamic model description. The sequence diagram (c) describes an operation call which transforms a system from an initial state (represented by the object diagram in (a)) to a successor state (represented by the object diagram in (b)). An equivalent representation of this behavior is shown in (d). Here, the respective states (snapshots) as well as the operation call (`marry`) are explicitly specified by a class diagram. As a consequence, the semantic of the description remains equivalent, but, as desired, only constructs of the basic model are applied.

The examples from above clearly show that a major cornerstone of the envisioned methodology, namely the transformation of various constructs into the basic model, is doable. Obviously, similar transformations are still left to be developed for other UML constructs and even other modeling languages. However, as initial feasibility studies documented in the next section confirm, the envisioned methodology is a promising step towards a holistic approach for system verification.

IV. Feasibility Studies

In order to demonstrate the applicability of the envisioned methodology, feasibility studies of the transformations described above have been conducted. To this end, by means of the existing tool USE [10], given UML/OCL models have been parsed, transformed into the basic model, and subsequently applied to a selected reasoning engine which solved a considered verification tasks on it.

Initial studies have been conducted on several designs which have been applied for benchmarking purposes before (e.g. in [1], [6]). By these studies, we were able to show that the basic model as introduced as above is sufficient to represent the respective models as well as the verification task for the considered cases. That is, verification tasks as conducted e.g. in [1], [6] can also be handled by the envisioned methodology and its basic model.

However, the restricted amount of constructs in the basic model (due to the desired atomicity) comes with a price: The respective instances are larger since e.g. the additional snapshot classes add further design elements in the model. As a result, the run-time of the reasoning engine increases. Some results of our studies illustrating this are presented in Table I. Here, reachability problems have been solved using the dedicated verification approach from [6] and the envisioned methodology, respectively, on the traffic light preemption benchmark proposed in [6]. The first column denotes the number of instantiated traffic light controllers, while the second and the fourth column (both denoted by *Size*) list the number of instantiated objects with respect to the used approach. As can be seen, the previous approach does not produce additional objects, whereas the envisioned methodology leads to overhead (e.g. due to the snapshot construction). That is, the instances of the envisioned methodology are significantly larger. As a result, it also takes longer to determine a result as shown in the third and fifth column (denoting the respective run-times in CPU seconds). However, the run-times are still reasonable and motivate a further investigation of the basic model as proposed.

Moreover, using the envisioned methodology enables for an easy exchange of reasoning engines. That, in turn, allows to accelerate the solving process just by choosing the most appropriate solver for the considered problem. In [6], the SMT solver *Boolector* [30] was applied (as was to obtain the results shown in the fifth column of Table I). However, due to the atomic structure of the basic models, interfaces to other reasoning engines can easily be added to the envisioned methodology. We did this for the SAT and SMT solving scheme of the *Z3* solver [31]. The resulting run-times are presented in the remaining columns of Table I. By doing this, even small improvements with respect to the dedicated verification method presented in [6] are possible.

TABLE I
EXPERIMENTAL EVALUATION

#	Previous approach [6]		Envisioned methodology			
	Size	Boolector	Size	Boolector	Z3 (SAT)	Z3 (SMT)
1	1	0.00	11	0.00	0.02	0.00
2	2	0.00	22	0.00	0.04	0.01
3	3	0.10	33	0.10	0.09	0.03
4	4	0.20	44	0.30	0.19	0.05
5	5	0.40	55	0.50	0.37	0.19
6	6	0.60	66	1.00	0.61	0.18
7	7	1.10	77	1.70	0.97	0.57
8	8	1.50	88	2.60	1.54	0.31
9	9	2.50	99	3.90	2.19	0.45
10	10	3.10	110	5.80	3.09	1.73
11	11	4.70	121	13.10	4.34	1.68
12	12	5.90	132	16.70	4.66	1.20
13	13	11.80	143	21.80	5.19	1.30
14	14	11.70	154	28.50	4.63	4.83
15	15	18.20	165	39.60	4.45	14.74
16	16	24.60	176	47.90	4.56	2.57

Overall, these feasibility studies encourage us in our motivation that the envisioned methodology and the basic model indeed cover features of existing verification approaches with an acceptable trade-off with respect to efficiency. At the same time, the applicability of the envisioned approach enabling a broader use with respect to the considered description means and verification tasks as outlined in Section II has been demonstrated. Surely, further transformations of constructs have to be developed and more intense studies need to be carried out. But the results available so far clearly motivate further research in this direction and, thus, articulate a promising solution for an emerging verification problem.

V. CONCLUSIONS

In this paper, we envisioned a flexible verification methodology for checking systems specified by means of modeling languages. The approach aims at supporting an arbitrary variation of constructs. New diagram types, verification tasks, and even reasoning engines should easily be added by just providing the respective transformation (instead of entirely re-implementing the overall approach). Combinations of more than one diagram (even of different types) should easily be performed by just composing the resulting basic models. Since the verification tasks shall be modeled using the same concepts as the specification, an integrated specification and verification environment is created. Furthermore, the envisioned methodology is aiming for a better performance since every verification task can be tackled by a range of different reasoning engines. As a result, major obstacles of today's approaches for model verification are addressed. By means of feasibility studies, the applicability of the envisioned approach has been shown. Future work is going to focus on the thorough development of model transformations for various constructs and modeling languages as well as a detailed consideration of the transformation of verification tasks.

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG) within the Reinhart Koselleck project DR 287/23-1.

REFERENCES

[1] M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler, "Verifying UML/OCL models using Boolean satisfiability," in *Design, Automation and Test in Europe*. IEEE Computer Society, Mar. 2010, pp. 1341–1344.

[2] F. Duran, M. Gogolla, and M. Roldan, "Tracing Properties of UML and OCL Models with Maude," in *Workshop Algebraic Methods in Model-based Software Engineering* . EPTCS 56, 2011, pp. 81–97.

[3] J. Cabot, R. Clarisó, and D. Riera, "Verification of UML/OCL Class Diagrams using Constraint Programming," in *IEEE Int'l. Conf. on Software Testing Verification and Validation Workshop*, Apr. 2008, pp. 73–80.

[4] B. Demuth and C. Wilke, "Model and Object Verification by Using Dresden OCL," in *Russian-German Workshop Innovation Information Technologies: Theory and Practice*. Technical University, July 2009, p. 81.

[5] D. Chiorean, M. Pasca, A. Cârcu, C. Botiza, and S. Moldovan, "Ensuring UML Models Consistency Using the OCL Environment," *Electr. Notes Theor. Comput. Sci.*, vol. 102, pp. 99–110, 2004.

[6] M. Soeken, R. Wille, and R. Drechsler, "Verifying Dynamic Aspects of UML Models," in *Design, Automation and Test in Europe*. IEEE Computer Society, Mar. 2011, pp. 1077–1082.

[7] J. Cabot, R. Clarisó, and D. Riera, "Verifying UML/OCL Operation Contracts," in *Integrated Formal Methods*, ser. Lecture Notes in Computer Science, M. Leuschel and H. Wehrheim, Eds., vol. 5423. Springer, Feb. 2009, pp. 40–55.

[8] A. Queralt and E. Teniente, "Reasoning on UML Class Diagrams with OCL Constraints," in *ER*, 2006, pp. 497–512.

[9] M. Gogolla, M. Kuhlmann, and L. Hamann, "Consistency, Independence and Consequences in UML and OCL Models," in *Tests and Proofs*. Springer, Jul. 2009, pp. 90–104.

[10] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-based specification environment for validating UML and OCL," *Science of Computer Programming*, vol. 69, no. 1-3, pp. 27–34, 2007, USE is available at http://sourceforge.net/apps/mediawiki/useocl/.

[11] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A Challenging Model Transformation," in *Int'l Conf. on Model Driven Engineering Languages and Systems*. Springer, Oct. 2007, pp. 436–450.

[12] M. Kuhlmann, L. Hamann, and M. Gogolla, "Extensive Validation of OCL Models by Integrating SAT Solving into USE," in *Int'l. Conf. on Objects, Models, Components, Patterns*, ser. Lecture Notes in Computer Science, vol. 6705. Springer, Jun. 2011, pp. 290–306.

[13] Z. Chen and D. Zhenhua, "Specification and Verification of UML2.0 Sequence Diagrams Using Event Deterministic Finite Automata," in *Int'l Conf. on Secure Software Integration Reliability Improvement Companion*, Jun. 2011, pp. 41–46.

[14] V. Lima, C. Talhi, D. Mouheb, M. Debbabi, L. Wang, and M. Pourzandi, "Formal Verification and Validation of UML 2.0 Sequence Diagrams using Source and Destination of Messages," *Electronic Notes in Theoretical Computer Science*, vol. 254, pp. 143–160, Oct. 2009.

[15] T. T. Dinh-Trong, S. Ghosh, R. B. France, M. Hamilton, and B. Wilkins, "UMLAnT: an Eclipse plugin for animating and testing UML designs," in *OOPSLA Workshop on Eclipse Technology eXchange*, Oct. 2005, pp. 120–124.

[16] R. Eshuis and R. Wieringa, "Tool support for verifying UML activity diagrams," *IEEE Trans. Software Eng.*, vol. 30, no. 7, pp. 437–447, 2004.

[17] V. Rafe, R. Rafeh, S. Azizi, and M. Miralvand, "Verification and Validation of Activity Diagrams Using Graph Transformation," in *Int'l Conf. on Computer Technology and Development*, vol. 1, Nov. 2009, pp. 201 –205.

[18] V. S. W. Lam, "A formalism for reasoning about UML activity diagrams," *Nordic J. of Computing*, vol. 14, pp. 43–64, Jan. 2007.

[19] C. Choppy, K. Klai, and H. Zidani, "Formal verification of UML state diagrams: a petri net based approach," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 1, pp. 1–8, 2011.

[20] C. Schwarzl and B. Peischl, "Static- and Dynamic Consistency Analysis of UML State Chart Models," in *Int'l Conf. on Model Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, vol. 6394. Springer, Oct. 2010, pp. 151–165.

[21] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language reference manual*. Essex, UK: Addison-Wesley Longman, Jan. 1999.

[22] T. Weilkiens, *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Feb. 2008.

[23] J. Warmer and A. Kleppe, *The Object Constraint Language: Precise modeling with UML*. Boston, MA, USA: Addison-Wesley Longman, Mar. 1999.

[24] M. Kyas, H. Fecher, F. S. de Boer, J. Jacob, J. Hooman, M. van der Zwaag, T. Arons, and H. Kugler, "Formalizing UML Models and OCL Constraints in PVS," *Electronic Notes in Theoretical Computer Science*, vol. 115, pp. 39–47, 2005.

[25] A. D. Brucker and B. Wolff, "The HOL-OCL Book," ETH Zurich, Tech. Rep. 525, 2006.

[26] B. Beckert, R. Hähnle, and P. Schmitt, *Verification of Object-Oriented Software: The KeY Approach*. Secaucus, NJ, USA: Springer, Oct. 2007.

[27] M. Cadoli, D. Calvanese, G. D. Giacomo, and T. Mancini, "Finite Model Reasoning on UML Class Diagrams Via Constraint Programming," in *AI*IA*, ser. Lecture Notes in Computer Science, R. Basili and M. T. Pazienza, Eds., vol. 4733. Springer, 2007, pp. 36–47.

[28] M. Gogolla and M. Richters, "Expressing UML Class Diagrams Properties with OCL," in *Advances in Object Modelling with the OCL*. Springer, 2001, pp. 86–115.

[29] I. Oliver and S. Kent, "Validation of Object Oriented Models using Animation," in *EUROMICRO Conf.* IEEE Computer Society, Sep. 1999, pp. 237–242.

[30] R. Brummayer and A. Biere, "Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays," in *Tools and Algorithms for Construction and Analysis of Systems*. Springer, Mar. 2009, pp. 174–177.

[31] L. M. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Tools and Algorithms for Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science, C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963. Springer, Apr. 2008, pp. 337–340.