CheckVML: A Tool for Model Checking Visual Modeling Languages

Ákos Schmidt and Dániel Varró

Budapest University of Technology and Economics
Department of Measurement and Information Systems
H-1521 Budapest, Magyar tudósok körútja 2.
varro@mit.bme.hu

Abstract. In the paper, we present a tool for model checking dynamic consistency properties in arbitrary well-formed instance models of any modeling language defined visually by metamodeling and graph transformation techniques. Our tool first translates such high-level specifications into a tool independent abstract representation of transition systems defined by a corresponding metamodel. From this intermediate representation the input language of the back-end model checker tool (i.e., SPIN in our case) is generated automatically.
Keywords: visual modeling languages, metamodeling, graph transformation, model checking, formal verification.

1 Towards a Formal Analysis of Modeling Languages in MDA

As the Model Driven Architecture (MDA) is becoming more and more widespread in the design process of IT systems, there is an increasing need for efficiently developing modeling languages and their model instances within a single modeling framework. For instance, UML itself (from version 2.0) is evolving into a family of modeling languages from a single and monolith language.

The definition of such modeling languages is frequently based on a combination of visual metamodeling techniques and well-formedness constraints expressed in the Object Constraint Language (OCL) that allow an object-oriented specification of the *static semantics* of the language. Since the current MOF standard does not provide an appropriate means to precisely specify the *dynamic operational semantics* of a language, many approaches (e.g., [1,5]) facilitate the use of graph transformation for that purpose. Graph transformation rules provide a visual and pattern based manipulation of the target user model fitting well to best engineering practices.

However, as the use of visual modeling techniques alone does not guarantee the correctness of a design, model checking tools (like SPIN [2]) are frequently used to mechanically analyze the functional correctness of system models based on UML statecharts (see e.g., [3]). Unfortunately, these approaches does not scale up well for ensuring consistency between multiple modeling languages as projecting modeling languages into a common semantic domain for verification purposes (as done, e.g., in [1]) can be difficult even for domain experts since input languages of model checker tools are very low-level. Moreover, a new tool has to be developed to extend the consistency framework for handling a new modeling language.

To overcome these problems, in the paper we present the CheckVML system, which provides tool support for the meta-level translation algorithm (presented in [4]) from high-level specification of modeling languages (based on metamodeling and graph transformation) into input languages of the model checker tools (currently implemented for SPIN [2]). As a result, the formal analysis of (well-formed instances of) a new modeling language can be highly automated and it does not require additional low-level and manual encoding efforts for enabling the use of model checker tools.

In the paper, we overview the visual specification of modeling languages and their well-formed user models in Sec. 2 with a small illustrating example. Section 3 introduces the basic architecture of a tool for model checking visual modeling languages.

2 Visual Specification of Modeling Languages

In our tool, the static structure of the modeling language is defined by metamodeling techniques while the dynamic operational semantics of the language is specified by graph transformation rules. Such descriptions will serve as the input of our transformation procedure to produce an equivalent transition system as result.

Models and metamodels Metamodeling is a term for capturing the evolution of user models and modeling languages uniformly, in a single visual modeling framework (in the form of UML class and object diagrams). The formal representation of such models and modeling languages can rely on the use of typed, and attributed graphs.

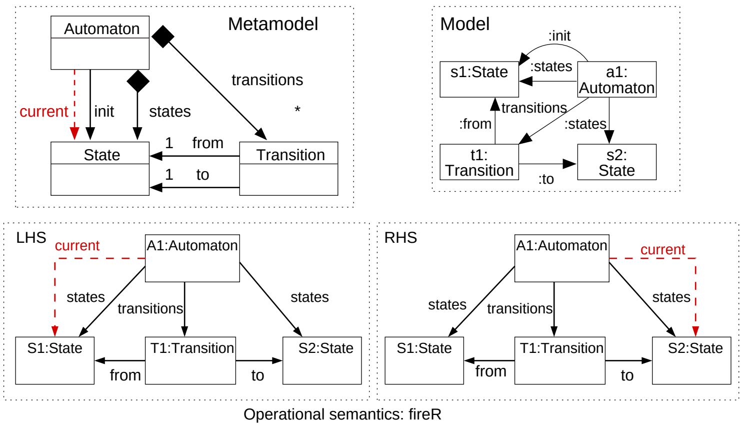

Fig. 1. Visual specification of modeling languages

Example 1. According to the metamodel of Fig. 1, a finite automaton consists of states and transitions. Each transition has a from state and a to state. The initial states of an automaton are marked with init and active states are marked with current associations.

A simple automaton instance a1 has two states (s1 and s2) and a transition (t1) between them (from s1 to s2). The initial state of a1 is s1, marked with an init link.

In the metamodel, we clearly separate dynamic components (i.e., those that can be modified by graph transformation rules like the current association in our case) from static ones by depicting them in dashed lines.

Graph transformation rules The specification of the dynamical operational semantics of modeling languages is defined on the meta-level by graph transformation rules. An application of a rule provides a pattern based manipulation on the target graph describing a legal transition in the state space of the modeled system.

A graph transformation rule is a triple $R = (Lhs, Neg, Rhs)$, where Lhs is the left-hand side graph (prescribing the preconditions), Rhs is the right-hand side graph (for the postconditions) and Neg graph contains the negative application conditions for the rule application.

Example 2. A simple graph transformation rule is depicted in the bottom part of Fig. 1, which shows the firing of a transition in a finite automaton. If S1 is the current active state in automaton A and a transition T exists from state S1 to an other state S2 (as described by Lhs) then the next active state in automaton A will be state S2 (see Rhs) when applying rule *fireR*.

3 CheckVML: A Tool for Model Checking Visual Modeling Languages

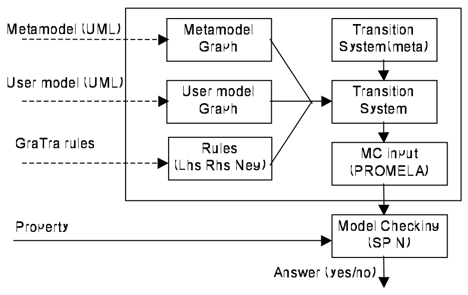

Fig. 2. The architecture of CheckVML

The CheckVML tool (see the architecture depicted in Fig. 2) aims at the formal verification of arbitrary visual modeling languages (like, such as UML , Petri nets, dataflow nets, etc.) by generating a model-level specification for the SPIN model checker. According to our optimization technique (in [4]), only dynamic concepts of a modeling language (i.e., those that may be modified by a graph transformation rule) are projected into the target transformation system in order to avoid state space explosion. The tool is written in Java in order to ensure platform independence.

- **Input:** The input of the tool (marked with dashed edges) consists of the *metamodel* of the modeling language, an *instance model*, and a set of *graph transformation rules*.
- **Output:** From these inputs our tool derives a semantically equivalent (model dependent) *transition system* following the guidelines of [4]. Transition systems are a common mathematical formalism that serves as the input specification of various model checker tools.

– **Tool independence:** Note that this intermediate transition system is defined by a corresponding metamodel, which provides a visual and tool independent specification of abstract mathematics that is easily understood by domain experts.
– **Model checking:** Finally, our tool generates a Promela description from the transition system which serves as the input for the SPIN model checker [2]. Supplied with a dynamic consistency property (in the form of an LTL formula), formal verification is carried out within SPIN.

Example 3. When feeding our sample finite automaton model as input together with its metamodel and graph transformation rule (see Fig. 1), CheckVML generates the following (extract of) Promela guarded command for applying rule *fireR* on a single matching.

```
if
:: current[a1][s1] -> current[a1][s1] = 0; current[a1][s2] = 1;
fi
```

Note that all the static concepts of the graph transformation rule are eliminated at compile time and state variables are only introduced for dynamic components according to our optimization technique.

4 Conclusion and Future Work

We presented a tool for automatically generating a Promela description from a meta-level visual specification of modeling languages composed of a metamodel, an instance model, and a set of graph transformation rules defining the operational semantics of the language. Simple case studies (including Petri nets and the verification benchmark of dining philosophers presented in [4]) have already been carried out to demonstrate the feasibility of our tool. In the future, we are aiming to apply our model checking tool for proving property preservation of model transformations between modeling languages within MDA.

References

1. G. Engels, R. Heckel, and J. M. Küster. Rule-based specification of behavioral consistency based on the UML meta-model. In M. Gogolla and C. Kobryn (eds.), *UML 2001: The Unified Modeling Language. Modeling Languages, Concepts and Tools*, vol. 2185 of *LNCS*, pp. 272–286. Springer, 2001.
2. G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, vol. 23(5):pp. 279–295, 1997.
3. D. Latella, I. Majzik, and M. Massink. Automatic verification of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, vol. 11(6):pp. 637–664, 1999.
4. D. Varró. Towards automated formal verification of visual modeling languages by model checking. *Journal of Software and Systems Modelling*, 2003. Submitted to the Special Issue on Graph Transformation and Visual Modelling Techniques.
5. D. Varró and A. Pataricza. Metamodeling mathematics: A precise and visual framework for describing semantics domains of UML models. In J.-M. Jézéquel, H. Hussmann, and S. Cook (eds.), *Proc. Fifth Intern. Conf. on the Unified Modeling Language – The Language and its Applications*, vol. 2460 of *LNCS*, pp. 18–33. Springer, Dresden, Germany, 2002.