Computer Languages, Systems & Structures 44 (2015) 72–88

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Intra- and interdiagram consistency checking of behavioral multiview models

Petra Kaufmann [a,*], Martin Kronegger [b,*], Andreas Pfandler [b,e,*], Martina Seidl [a,c,*], Magdalena Widl [d,*]

[a] *Business Informatics Group, TU Wien, Karlsplatz 13, 1040 Wien, Austria*
[b] *Database and Artificial Intelligence Group, TU Wien, Karlsplatz 13, 1040 Wien, Austria*
[c] *Inst. f. Formal Models and Verification, JKU Linz, Altenbergerstr. 69, 4040 Linz, Austria*
[d] *Knowledge-Based Systems Group, TU Wien, Karlsplatz 13, 1040 Wien, Austria*
[e] *School of Economic Disciplines, Univ. Siegen, A.-Reichweinstr. 2, 57076 Siegen, Germany*

A R T I C L E I N F O

Article history:
Received 20 March 2015
Received in revised form
5 June 2015
Accepted 7 August 2015
Available online 18 August 2015

Keywords:
Multiview modeling
Unified modeling language
Consistency checking
SAT encodings

A B S T R A C T

Multiview modeling languages like UML are a very powerful tool to deal with the ever increasing complexity of modern software systems. By splitting the description of a system into different views—the diagrams in the case of UML—system properties relevant for a certain development activity are highlighted while other properties are hidden. This multiview approach has many advantages for the human modeler, but at the same time it is very susceptible to various kinds of defects that may be introduced during the development process. Besides defects which relate only to one view, it can also happen that two different views, which are correct if considered independently, contain inconsistent information when combined. Such inconsistencies between different views usually indicate a defect in the model and can be critical if they propagate up to the executable system.

In this paper, we present an approach to formally verify the reachability of a global state of a set of communicating UML state machines, i.e., we present a solution for an intradiagram consistency checking problem. We then extend this approach to solve an interdiagram consistency checking problem. In particular, we verify whether the message exchange modeled in a UML sequence diagram conforms to a set of communicating state machines.

For solving both kinds of problems, we proceed as follows. As a first step, we formalize the semantics of UML state machines and of UML sequence diagrams. In the second step, we build upon this formal semantics and encode both verification tasks as decision problems of propositional logic (SAT) allowing the use of efficient SAT technology. We integrate both approaches in a graphical modeling environment, enabling modelers to use formal verification techniques without any special background knowledge. We experimentally evaluate the scalability of our approach.

* Corresponding authors.
E-mail addresses: kaufmann@big.tuwien.ac.at (P. Kaufmann), kronegger@dbai.tuwien.ac.at (M. Kronegger), pfandler@dbai.tuwien.ac.at (A. Pfandler), martina.seidl@jku.at (M. Seidl), widl@kr.tuwien.ac.at (M. Widl).

http://dx.doi.org/10.1016/j.cl.2015.08.003
1477-8424/© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

A major difference between traditional software engineering and model-driven engineering (MDE) [5] lies in the nature of the core development artifacts. These artifacts, which in traditional software engineering comprise mainly textual code, are represented by (visual) software models in MDE. Often software models are expressed in multiview modeling languages like the *Unified Modeling Language* (UML) [25], where a focused view on specific aspects (e.g., behavioral or structural aspects) of the system under consideration is given. The goal of MDE is to leverage the abstraction power offered by software models to deal with the complexity of modern software systems [3], and to further exploit the models to automatically generate executable code with little or no intervention of a human developer [28].

The increasing valorization of software models imposes stronger demands and expectations on their correctness. In their role as core development artifacts, software models are increasingly sensitive to the impact of evolution and therefore more exposed to the introduction of errors [13]. Especially the abstraction power of multiview modeling languages as offered by UML bears the danger of introducing inconsistencies into the model under development [22].

Inconsistent software models can be the root of severe problems if they are employed for automatic code generation because inconsistencies can propagate to the executable system and result in serious errors in the application. Hence, if the diagrams do not complement each other in a consistent manner, then the benefits of multiview modeling will decrease or even vanish [28]. Due to the multiview nature and the size of software models, inconsistencies are often hard to spot for a human developer. Especially when the models are not directly executable or when no simulation environment is available, testing and debugging is difficult. Here, formal verification methods can help to ensure that the models fulfill intradiagram and interdiagram consistency criteria, i.e., the consistency is ensured within one diagram and between different diagrams, respectively.

In this paper, we first consider the following *intradiagram consistency checking problem*: For a set of communicating state machines, which describe the internal behavior of objects, we check if it is consistent to assume that a specific system configuration, i.e., a (partial) global state, is reachable from the initial state. If the answer is affirmative, then the respective execution path is returned. Hence, the (partial) global states are test cases, asserting allowed or forbidden system configurations.

We then extend this intradiagram consistency checking problem to an *interdiagram consistency checking problem* of state machines and sequence diagrams. Sequence diagrams focus on interaction scenarios between different instances of classes and the respective state machines. These scenarios model either required or forbidden message exchange. Our approach verifies whether the communication described by a sequence diagram can be executed by a given set of state machines in a state reachable from the initial state. If a forbidden sequence of messages can be executed, then a concrete communication trace is returned. If a sequence of messages is not possible although according to the sequence diagram it should be, then a reason for the failure is given. On this basis, inconsistencies introduced during the evolution of a model cannot only be discovered easily, but also be corrected immediately. Hence, sequence diagrams are test cases describing desired or undesired behavior of the state machines. With our approach the test cases can be evaluated even if no execution environment for the state machines is available.

A crucial ingredient for an implementation of the above-mentioned consistency checks is a well-defined, formal semantics of the diagrams types that are to be verified. Therefore, we first introduce a formal semantics for UML state machines and UML sequence diagrams, and then we propose an approach to solve the consistency problems based on a reduction to the satisfiability problem of propositional logic (SAT) [4]. For SAT powerful solvers are available, which can successfully be used out of the box in many applications.

This paper is structured as follows. First, we review related approaches in Section 2. Then we motivate this work with a concrete example in Section 3 and informally explain the modeling language concepts relevant for this work. In Section 4 we give a concise formal problem definition. To this end, we formally describe sequence diagrams and state machines along with their interplay. Further, we introduce the notion of global state reachability and sequence consistency, which are essential for our problem definitions. These problem definitions allow us to come up with a translation of the consistency checking problems to propositional formulas, which can be handed to a SAT solver (Section 5). In Section 6 we discuss the implementation based on the Eclipse Modeling Framework and in Section 7 we present a detailed evaluation on randomly generated and on crafted models. Finally, we conclude with an outlook on future work.

This paper is an extended and revised version of our SLE 2014 [17] paper. Besides details on the technical realization and further experiments, we present the complete workflow of our verification framework. This includes enhancements of the global state checking approach, which was presented at the MoDeVVa 2013 Workshop [16].

2. Related work

We consider two different streams of work related to our approach. On the one hand, we review literature on reachability checking for state machines and on the other hand we give an overview on approaches for consistency checking between state machines and sequence diagrams.

Reachability checking: Several works have been presented which deal with the transformation of UML state machines to input languages of model checkers (see for example [1,9,18,21,24]). These languages provide high-level constructs to model