Information and Software Technology 56 (2014) 821–838

Contents lists available at ScienceDirect

Information and Software Technology

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

Formal verification of static software models in MDE: A systematic review

Carlos A. González [*], Jordi Cabot

AtlanMod Research Group, École des Mines de Nantes – INRIA, LINA, 4, Rue Alfred Kastler, F-44307 Nantes Cedex 3, France

A R T I C L E   I N F O

Article history:
Received 5 July 2013
Received in revised form 17 February 2014
Accepted 12 March 2014
Available online 19 March 2014

Keywords:
MDE
Formal verification
OCL
Systematic literature review

A B S T R A C T

Context: Model-driven Engineering (MDE) promotes the utilization of models as primary artifacts in all software engineering activities. Therefore, mechanisms to ensure model correctness become crucial, specially when applying MDE to the development of software, where software is the result of a chain of (semi)automatic model transformations that refine initial abstract models to lower level ones from which the final code is eventually generated. Clearly, in this context, an error in the model/s is propagated to the code endangering the soundness of the resulting software. Formal verification of software models is a promising approach that advocates the employment of formal methods to achieve model correctness, and it has received a considerable amount of attention in the last few years.
Objective: The objective of this paper is to analyze the state of the art in the field of formal verification of models, restricting the analysis to those approaches applied over static software models complemented or not with constraints expressed in textual languages, typically the Object Constraint Language (OCL).
Method: We have conducted a Systematic Literature Review (SLR) of the published works in this field, describing their main characteristics.
Results: The study is based on a set of 48 resources that have been grouped in 18 different approaches according to their affinity. For each of them we have analyzed, among other issues, the formalism used, the support given to OCL, the correctness properties addressed or the feedback yielded by the verification process.
Conclusions: One of the most important conclusions obtained is that current model verification approaches are strongly influenced by the support given to OCL. Another important finding is that in general, current verification tools present important flaws like the lack of integration into the model designer tool chain or the lack of efficiency when verifying large, real-life models.

Contents

[*] Corresponding author. Tel.: +33 (0)2 51 85 82 16.
E-mail addresses: carlos.gonzalez@mines-nantes.fr (C.A. González), jordi.cabot@mines-nantes.fr (J. Cabot).

http://dx.doi.org/10.1016/j.infsof.2014.03.003
0950-5849/© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Model-driven Engineering (MDE) is an approach that promotes the utilization of models as primary artifacts in all software engineering activities. In a MDE-based software development process, the software is not coded by hand, but by designing and creating a number of models to be successively and (semi)automatically transformed into more refined models, and eventually into code comprising the new software system. When MDE approaches are used to develop complex systems, the complexity of models involved increases, thus turning creating and editing them into error-prone tasks. This complexity endangers the MDE development process and the soundness of the resulting software.

Ensuring software correctness is not a new challenge, though. On the contrary, it is an old challenge that software engineers continue to struggle with [1]. Thanks to the efforts made by the research community, different trends have appeared to try to address the problem. One of them is software verification, which comprises those approaches based on the utilization of formal methods and formal analysis techniques to prove software correctness. The word "formal" states that these methods are based on mathematical theories. One of the major advantages of using formal methods to verify software is to avoid the introduction of imprecision or ambiguity in the process. In these approaches, it is typical to proceed by going through two different stages. In the first one, the formalization stage, the problem to be solved is represented using one of these mathematical theories or formalisms. In the second one, the reasoning stage, the resolution of the formalized problem is addressed by utilizing tools specialized in reasoning over the chosen formalism.

Considering the problems that model complexity has brought into MDE-based software development processes and the advantages of the utilization of formal methods when verifying software, it comes as no surprise that a lot of effort has been made in studying how to apply formal methods and formal analysis techniques to ensure model correctness. This has obviously led to a significant amount of research results.

While working on [2], we had the opportunity of making a first analysis of some of this research. It was hard for us to grasp a clear view, each work using their own terminology to describe the problem, in some cases, borrowed from what we considered were different fields, like consistency checking or software validation. Because of this, we considered that, in order to check whether this initial impression of ours was right, a deeper analysis was needed. In this regard, although some systematic literature reviews had been conducted to analyze notions like model quality [3] or the consistency between models [4], we did not know of the existence of studies devoted to the analysis of the utilization of formal methods and formal analysis techniques to ensure model correctness.

It is also true, though, that addressing this type of exhaustive analysis in its full generality, and without considering factors like model diversity, is probably inadequate, leading for example to the comparison of approaches that use models of very different nature, and impractical, since it would probably produce cumbersome results. Because of these factors, the systematic literature review presented in this paper focuses on providing an exhaustive analysis and comparison of the research initiatives done in the field of formal verification of static models, only. The reason for this is that static models are arguably the models more commonly adopted at the time of describing the specification of a software system.

Static models, also commonly referred to as structural models, are those models used to represent, totally or partially, the structure of a system design, that is, those models that show a time independent view of the system. Regarding this, the most popular static model is the UML[1] (Unified Modeling Language) class diagram, although in this study, other models that fall outside of the UML umbrella, like for example entity-relationship models or models developed by using Domain Specific Modeling Languages (DSMLs), were also covered. Also related to the type of models covered in this study, it is important to notice that since UML class diagrams exhibit certain limitations to represent precisely detailed aspects of a system, they are often used in combination with OCL[2] (Object Constraint Language). OCL provides additional capabilities to enrich models, like for example, a mechanism for the definition of integrity constraints that the instances of a model must hold. The analysis of how OCL is supported by the existing model verification approaches was also covered as part of this study. It is also worth remarking that model correctness refers to the ability of the model under analysis to satisfy one or more correctness properties (like for example satisfiability). Obviously, another important part of this study is to analyze what correctness properties are supported by every verification approach.

More specifically, this study addressed the following research questions:

- RQ1: What are the typical phases in model verification approaches?

[1] http://www.omg.org/spec/UML/.
[2] http://www.omg.org/spec/OCL/.