VU VRIJE UNIVERSITEIT AMSTERDAM

VU Research Portal

Bringing Model Checking Closer To Practical Software Engineering

Remenska, D.

2016

document version
Publisher's PDF, also known as Version of record

Link to publication in VU Research Portal

citation for published version (APA)
Remenska, D. (2016). *Bringing Model Checking Closer To Practical Software Engineering*. [PhD-Thesis - Research and graduation internal, Vrije Universiteit Amsterdam].

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

E-mail address:
vuresearchportal.ub@vu.nl

Download date: 04. Nov. 2024

Software grows in size and complexity, making it increasingly challenging to ensure that it behaves correctly. This is especially true for distributed systems, where a multitude of components are running concurrently, making it difficult to anticipate all the possible behaviors emerging in the system as a whole. Certain design errors, such as deadlocks and race-conditions, can often go unnoticed when testing is the only form of verification employed in the software engineering life-cycle. Even when bugs are detected in a running software, revealing the root cause and reproducing the behavior can be time consuming (and even impossible), given the lack of control the engineer has over the execution of the concurrent components, as well as the number of possible scenarios that could have produced the problem. This is especially pronounced for large-scale distributed systems such as the Worldwide Large Hadron Collider Computing Grid.

Formal verification methods offer more rigorous means of determining whether a system satisfies certain behavioral requirements, and one highly effective verification technique is model checking. Model checking is a mathematically-rooted algorithmic procedure, nowadays automated by many actively-maintained and mature tools. The drawback, however, is the necessity to be proficient in formal language notations such as process algebras and temporal logics, essential for describing a model of the system and the behavioral requirements to be verified by these tools.

In this thesis we address the question of how to integrate model checking into the common software development cycle of realistic-scale, distributed, data-driven software, by automating the aspects that require formal methods expertise. For this purpose, we chose a formalism powerful enough for modeling and addressing design errors that are common for such software. We are motivated by the distributed grid framework called DIRAC, developed and used by a physics community at the European Organization for Nuclear Research (CERN). We examine the feasibility of using the mCRL2 language, by systematically abstracting the source code and modeling the behavior of two DIRAC subsystems. The case studies indicate that mCRL2 has the necessary concurrency, data abstraction and manipulation mechanisms to faithfully model the subsystems. By simulating, visualizing, and model checking the resulting models with the mCRL2 toolset, we were also able to gain a better insight into the system behavior, and replaying the counter-example traces helped in localizing the detected problems. Although some of the faulty behaviors already manifested in real life before the subsystems were formally modeled, they were not localized within the timespan of the modeling and verification we performed.

Manually constructing a formal model based a software implementation can be time consuming. In addition to the risk of making modeling mistakes, in order to be useful, the formal model must be kept up to date with the continuous software updates. Software implementations contain too many language-specific details to serve as footprints for deriving formal models. In software engineering, higher-level visual designs are created for communicating and validating the requirements, before the implementation and testing takes place. UML is generally accepted as a visual modeling language for this purpose. We present

a transformation approach for automatically deriving mCRL2 formal models, based on UML sequence and activity diagrams. We discuss the semantic choices we made with respect to some ambiguities in the official semi-formal UML semantics, and compare our approach to existing ones, in the context of a well-known classification. The transformation preserves the object-oriented structure of the system, facilitating a straightforward round-trip approach in which the verification counter-examples can also be visually presented as sequence diagrams. To provide some empirical evidence and confidence in the correctness of the translation, we apply the tool-supported approach on DIRACs new workload system functionality. We discover the root cause of a logical flaw leading to no-progress, which had been observed earlier in the testing phase of this functionality.

In model checking, behavioral properties must be expressed as formulas in temporal logic. The mCRL2 toolset requires the use of the modal μ-calculus for this; a very expressive logic, but not very intuitive nor accessible. Behavioral requirements for software are typically expressed in natural language, and as such can be subjects to different interpretations. In the context of the main research question, to bring the process of correctly eliciting behavioral properties closer to software engineers, we introduce a property assistant tool - PASS, as part of a UML-based front-end to the mCRL2 toolset. The tool provides assistance to non-experts in eliciting properties which capture the required behavior precisely and unambiguously. It is based on a well-known property pattern classification, which we extend with new pattern variations for the event-based modal μ-calculus. Besides a μ-calculus formula, the tool outputs a natural language summary, and a UML sequence diagram depicting the property. In addition, for a subclass of properties, PASS automatically generates monitoring structures which can be used for a (potentially) more efficient property-driven runtime verification. We demonstrate the usage of PASS on DIRACs behavior.

The property pattern classification is based on a large literature survey of how specification formalisms are used in practise. Since we did not encounter any μ-calculus formulas in the collection repository, we were curious whether its usage differs significantly from other formalisms. We surveyed 25 published works that use μ-calculus to express system properties from different domains. From the 178 properties in the survey, our pattern extensions improve the standard patterns classification coverage by 10compared to the more commonly used logics (like LTL- Linear Temporal Logic, en CTL - Computation Tree Logic), is able to capture certain property patterns which either LTL or CTL cannot. We observed a distribution of patterns which is rather consistent with similar surveys focusing on other formalisms. The results also indicate that subtle mistakes can easily be made when constructing temporal logic formulas manually.

As one possible future direction, it would be interesting to extend this work beyond discovery of behavioral problems, in particular in the area of performance assessment of UML models. Although this type of analysis is a primary concern for the domain of real-time and embedded systems, valuable insight can also be gained about communication costs, delays, and various bottlenecks in distributed systems.