Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Delta-oriented model-based integration testing of large-scale systems[☆]

Malte Lochau[a], Sascha Lity[b,*], Remo Lachmann[c], Ina Schaefer[c], Ursula Goltz[b]

[a] *TU Darmstadt, Real-Time Systems Lab, Germany*
[b] *TU Braunschweig, Institute for Programming and Reactive Systems, Germany*
[c] *TU Braunschweig, Institute of Software Engineering and Automotive Informatics, Germany*

ARTICLE INFO

Article history:
Received 19 November 2012
Received in revised form 4 November 2013
Accepted 17 November 2013
Available online 23 November 2013

Keywords:
Large-scale systems
Model-based testing
Regression testing
Variable software architectures

ABSTRACT

Software architecture specifications are of growing importance for coping with the complexity of large-scale systems. They provide an abstract view on the high-level structural system entities together with their explicit dependencies and build the basis for ensuring behavioral conformance of component implementations and interactions, e.g., using model-based integration testing. The increasing inherent diversity of such large-scale variant-rich systems further complicates quality assurance. In this article, we present a combination of architecture-driven model-based testing principles and regression-inspired testing strategies for efficient, yet comprehensive variability-aware conformance testing of variant-rich systems. We propose an integrated delta-oriented architectural test modeling and testing approach for component as well as integration testing that allows the generation and reuse of test artifacts among different system variants. Furthermore, an automated derivation of retesting obligations based on accurate delta-oriented architectural change impact analysis is provided. Based on a formal conceptual framework that guarantees stable test coverage for every system variant, we present a sample implementation of our approach and an evaluation of the validity and efficiency by means of a case study from the automotive domain.

1. Introduction

Modern software systems are increasingly large-scaled. They often exist in many different variants in order to adapt to their environment context. Additionally, they evolve over time into different versions in order to satisfy changing user requirements. This diversity in space and time causes high complexity during system design and implementation as well as for quality assurance. Software architecture (SA) specifications are of growing importance for coping with these variant-rich large-scale software systems. SAs provide an abstract view on the high-level structural system entities together with their explicit interdependencies which alleviates complexity during system development. This decreases development complexity by allowing a hierarchical decomposition of the overall system into smaller subsystems and their components. For quality assurance, an architectural design builds the basis for ensuring behavioral conformance of component implementations and interactions w.r.t. architectural specifications, e.g., using model-based component and integration testing.

The adoption of SA testing approaches to variant-rich software systems faces the challenge to ensure correctness of component implementations and interactions for any possible system variant which is, in general, infeasible due to their high number. Even if every system variant could be tested in isolation, this is very inefficient because of the commonality between the variants that is checked over and over again.

To counter this problem, we combine architecture-driven model-based testing principles and regression-based testing strategies for efficient, yet comprehensive conformance testing on the component and on the integration test level. We model the expected behavior of a system by its architectural specification in terms of components and connectors. The intra-component behavior is captured by state machines. However, applying the corresponding model-based testing techniques also to component integration testing on the basis of a parallel composition of those component state machines is, in general, impracticable for large-scale systems due to the well-known state-explosion-problem. Instead, component interaction scenarios to be tested are explicitly specified by message sequence charts as proposed, e.g., in Briand et al. (2012). In order to express system diversity, we apply the principles of delta modeling to the system description on the

☆ This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution.

* Corresponding author. Tel.: +49 531 391 3276.
E-mail address: lity@ips.cs.tu-bs.de (S. Lity).

architectural as well as on the component level. Delta modeling (Clarke et al., 2010; Schaefer et al., 2010, 2011) is a modular, yet flexible way to capture variability in time and in space by explicitly specifying the changes between system variants. In this way, we obtain a concise description of the commonality and variability of the expected behavior between system variants. Thereupon, we are able to automatically derive a *regression delta* explicitly capturing differences among arbitrary system variants/versions. In contrast to classical regression testing requiring fine-grained, therefore, expensive tracings of changes (Engström et al., 2008), e.g., by means of model differencing (Treude et al., 2007; Kelter and Schmidt, 2008), the regression delta allows us to efficiently reason about preplanned test artifact reuse among system versions/variants at all testing levels in a sound and uniform way. We further use this delta-oriented representation to reduce retesting common behaviors of system variants by relying on the principles of regression testing. The derivation of the regression testing obligations between two system variants is based on an impact analysis of the architectural and behavioral changes specified in the respective regression delta.

The contribution of this work is twofold: (1) we introduce delta-oriented architecture test modeling to achieve the systematic reuse of common test artifacts between different system variants and/or versions and (2) we apply delta-oriented regression planning for the systematic evolution of variable test artifacts among different software variants and/or versions. This article extends our previous work on delta-oriented component testing (Lochau et al., 2012) by leveraging the idea of delta-oriented test modeling and test artifact evolution to the integration test level and combines delta-oriented component and scalable integration testing of variant-rich software systems into one unified framework.

We illustrate and evaluate our approach in a prototypical implementation by means of a case study from the automotive domain, a simplified *Body Comfort System* (BCS). Its original version was proposed by Müller et al. (2009) in cooperation with an industrial partner and was enhanced to a variant-rich system by Oster et al. (2011). The BCS comprises a number of standard and optional functions. All BCS include by default a *Human Machine Interface* (HMI) as point of interaction with a driver, an electric *Exterior Mirror* (EM) with optional heating functionality, either an *Automatic Power Window* (AutoPW) or a *Manual Power Window* (ManPW), and a *Finger Protection* (FP) blocking the window movement when a finger is clamped in a window. Optionally, a BCS can consist of a *Remote Control Key* (RCK) enabling the locking/unlocking of the car as well as the controlling of the window movement, an *Alarm System* (AS) with optional *Interior Monitoring* (IM), a *Central Locking System* (CLS) with optional *Automatic Locking* (AL) when the car is driving, and for some functions like the CLS *Status LEDs* (LED) indicating the activation/inactivation of the corresponding function. The planned combination of those functional variants results in 11,616 possible variants of the BCS.

This article is structured as follows: In Section 2, we present the foundations for delta-oriented model-based testing. In Section 3, we describe our integrated framework for model-based component and integration testing. In Section 4, we apply the principles of delta modeling to obtain variable test models both on the component and the integration level. In Section 5, we show how to evolve the test artifacts between system variants and/or versions at all test levels based on the derivation of regression deltas and, thereupon, we describe an incremental testing work flow incorporating a systematic evolution of test artifacts. In Section 6, we present our prototypical tool chain. In Section 7, we describe the case study design and, thereupon, we discuss the results of our evaluation in Section 8. We compare our approach to related work in Section 9 and conclude in Section 10.

2. Fundamentals of delta-oriented model-based testing

In this section, we describe the general concepts of our incremental model-based testing approach for variant-rich large-scale (software) systems based on delta-oriented test model specifications. This generalized framework is later instantiated to guarantee efficient, yet reliable test coverage at both component as well as integration level.

2.1. Foundations of model-based testing

Model-based testing aims at the automation of black-box testing processes (Utting and Legeard, 2007). The fundamental test artifacts involved in those model-based testing processes for a single software system under test together with their interrelations are depicted in Fig. 1(a). A *test model tm* specifies the intended behaviors of the software implementation under test *sut*. The (partial) verification of the behavioral conformance of the *sut* to the test model *tm* is performed by selecting a finite set of *test cases*, i.e., representative executions of the software system extracted from the test model (De Nicola, 1987). The *sut* passes the experimental execution of a test case *tc* if it reacts as intended, i.e., as specified in the corresponding test model *tm*. Functional test cases usually define a sequence of controllable input stimuli to be injected into the software system under test, together with a corresponding sequence of system outputs stating the reactions expected for those inputs.

A test case *tc* is *valid* for a test model *tm* if it conforms to a behavior as specified by the test model. By $TC(tm)$ we refer to the set of all valid test cases defined by the test model *tm*. In general, this set contains an infinite number of test cases and the corresponding execution sequences are of potentially infinite length. The selection of a finite set of *representative* test cases of finite length into a *test suite* $TS \subseteq TC(tm)$ is usually based on *adequacy criteria* measuring the quality of test suites (Utting and Legeard, 2007). For instance, a model-based *coverage criterion CC* applied to a test model *tm* selects a finite subset $CC(tm) = TG \subseteq TG(tm)$ of *test goals*, i.e., particular structural elements occurring in the test model. By $TG(tm)$ we refer to the set of all possible test goals within test model *tm*.

For a test suite *TS* to satisfy a coverage criterion, each selected test goal $tg \in TG$ is to be *covered*, i.e., traversed by at least one test case $tc \in TS$. Correspondingly, we have an *n*-to-*m* correspondence

$$covers_{CC} \subseteq TC(tm) \times TG(tm)$$

between test cases $tc \in TS \subseteq TC(tm)$ and the set of test goals selected for a criterion *CC*. Please note that we may omit the index *CC* in the following if not relevant. These abstract notions for model-based testing are independent of the actual representation of test models, test cases, test goals, etc. Their concrete instantiation depends on the modeling formalism, the system level, and the development phase to be addressed by the corresponding testing campaigns (Utting and Legeard, 2007). As illustrated in Fig. 2, different kinds of test models and corresponding test artifacts are usually applied depending on the testing stage and the software units under consideration. For our architecture-based approach to model-based testing of single components, as well as their integration at the architectural level, we consider the following, mutually interrelated models:

- *Architecture model.* The decomposition of the overall software system into components and the specification of communication relationships between those components are specified in a high-level structural system description based on component-connector abstractions. We use those architecture models as a basis for planning component and integration testing of large-scale systems.