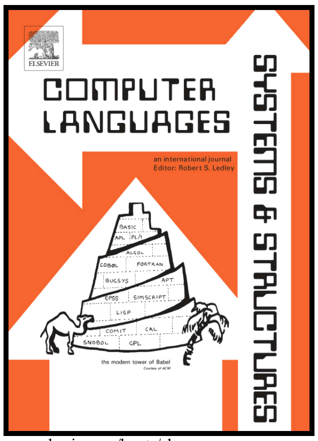
A Text-Based Visual Notation for the Unit Testing of Model-Driven Tools

Daniel Strüber, Felix Rieger, Gabriele Taentzer



PII: S1477-8424(16)30027-6
DOI: http://dx.doi.org/10.1016/j.cl.2016.08.004
Reference: COMLAN231

To appear in: *Computer Language*

Received date: 10 February 2016
Accepted date: 13 August 2016

A Text-Based Visual Notation for the Unit Testing of Model-Driven Tools

Daniel Strüber, Felix Rieger, Gabriele Taentzer

Philipps-Universität Marburg, Germany,
{strueber, riegerf, taentzer}@informatik.uni-marburg.de

Abstract. During the unit testing of model-driven tools, a large number of models and test classes needs to be managed and maintained. Typically, some of these artifacts are specified manually, some are generated automatically. Existing approaches to test management rely on the available visual and textual modeling notations. As these notations are not tailored to unit testing, distinct maintainability trade-offs arise.
In this paper, we propose a notation that aims to combine the benefits of visual and text-based approaches. The notation is at the same time visual and text-based, as it uses ASCII characters to emulate the familiar graphical notations. In our evaluation based on real models, we identify problematic model shapes challenging the scalability our notation, while finding that it is well-suited to capture typical test models.

1 Introduction

In Model-Driven Engineering, software engineers employ a large variety of tools to specify, transform, and manage models. Such *model-driven tools*, like all software artifacts, are routinely affected by software defects.

A standard practice to detect and resolve software defects is *unit testing*, the process of testing individual parts of a system to determine if they comply with a behavior specification [1]. In the context of model-driven tools, behavior is usually defined in relationship to the models processed by the tool. Therefore, a minimal unit test contains a model together with some test code that feeds the model as input to the tool. A growing body of work focuses on automating the creation of such unit tests to enable a high coverage of the involved code parts and meta-models [2 – 4]. As a result, a test suite typically contains a large number of models, some of them specified manually, some created automatically [5].

An important concern of unit tests just starting to attract attention is their maintainability. As pointed out in a recent developer study by Daka and Fraser, *"even when automatically generated, a unit test may be integrated into the regular code base, where it needs to be manually maintained like any other code."*[6] Several issues contribute to this concern. First, to fix a bug uncovered during testing, maintainers need to understand the test case eliciting the bug. Second, developers use tests as usage examples to understand a system's behavior. Notably, coverage and maintainability are not necessarily conflicting goals: Daka et al. found that readable unit tests can be generated without loss of coverage [7].