Journal of Visual Languages and Computing 24 (2013) 441–462

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

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

Model-driven rapid prototyping with programmed graph transformations ☆

Anthony Anjorin [1], Karsten Saller *,[2], Ingo Reimund, Sebastian Oster, Ivan Zorcic, Andy Schürr

Real-Time Systems Lab, Technische Universität Darmstadt, Germany

ARTICLE INFO

Available online 30 August 2013

Keywords:
Rapid prototyping
Programmed graph transformations
Metamodelling
Software product lines
Model-driven testing

ABSTRACT

Modern software systems are constantly increasing in complexity and supporting the *rapid prototyping* of such systems has become crucial to check the feasibility of extensions and optimizations, thereby reducing risks and, consequently, the cost of development. As modern software systems are also expected to be reused, extended, and adapted over a much longer lifetime than ever before, ensuring the *maintainability* of such systems is equally gaining relevance.

In this paper, we present the development, optimization and maintenance of MoSo-PoLiTe, a framework for *Software Product Line* (*SPL*) *testing*, as a novel case study for rapid prototyping via metamodelling and programmed graph transformations.

The first part of the case study evaluates the use of programmed graph transformations for optimizing an existing, hand-written system (MoSo-PoLiTe) via rapid prototyping of various strategies. In the second part, we present a complete re-engineering of the hand-written system with programmed graph transformations and provide a critical comparison of both implementations.

Our results and conclusions indicate that metamodelling and programmed graph transformation are not only suitable techniques for rapid prototyping, but also lead to more maintainable systems.

1. Introduction

Modern software systems are constantly increasing in size and complexity and it has become crucial to develop appropriate techniques to cope with the challenge of developing such systems at a reasonable cost. *Rapid prototyping* can be used to check if an extension or optimization of a system is feasible and performs as expected. As this is accomplished at a comparably low cost, the risk of making wrong decisions is avoided early in the development process, thus reducing the effective cost of software development.

As modern software systems are also expected to be reused, extended and adapted over a much longer lifetime than ever before, ensuring the *maintainability* of such systems is equally gaining relevance.

In this paper, we present a novel case study for metamodelling and programmed graph transformations and show with our results that our model-driven approach is not only suitable for rapid prototyping but also leads to a more maintainable system.

☆ This paper has been recommended for acceptance by Shi Kho Chang.

* Corresponding author. Tel.: +49 6151 16 3776.
E-mail addresses: anjorin@es.tu-darmstadt.de (A. Anjorin), saller@es.tu-darmstadt.de (K. Saller), reimund@es.tu-darmstadt.de (I. Reimund), oster@es.tu-darmstadt.de (S. Oster), zorcic@es.tu-darmstadt.de (I. Zorcic), schuerr@es.tu-darmstadt.de (A. Schürr).

[1] Supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School of Computational Engineering at TU Darmstadt.

[2] Supported by the German Research Foundation (DFG) in the Collaborative Research Center (SFB) 1053 "Multi-Mechanism-Adaptation for the Future Internet".

1045-926X/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.jvlc.2013.08.001

For our case study, we investigate MoSo-PoLiTe (Model-Based Software Product Line Testing) [8], a framework for Software Product Line (SPL) testing that combines and applies combinatorial testing and model-based testing to SPL feature models [9].

In the MoSo-PoLiTe approach, a feature model that describes the variability in an SPL as a tree of interrelated features [9], is converted to a Constraint Satisfaction Problem (CSP) via a series of semantics preserving transformation rules that flatten the feature tree appropriately [10]. As the sheer number of possible product configurations is impossible to test with classical approaches, a representative subset of product configurations is determined using a combinatorial criterion, e.g., that the subset must cover all valid pairwise combinations of all features.

This task equates to solving the CSP derived from the SPL using well-known approaches such as *forward checking* with some extensions. Details concerning how the chosen subset of products to be tested can be mapped to concrete test cases using a test model are discussed in [11].

As the flattening transformation that converts the feature model to a CSP is by no means unique and can be varied and optimized for a concrete CSP solver, *rapid prototyping* techniques can be applied to test and evaluate different optimization strategies.

An optimization strategy is, for example, to create redundant constraints that do not change the semantics of the CSP but lead to a reduction of the search space for valid combinations. This basically results in a trade-off of memory (redundant constraints and annotations in the flattened tree) for efficiency (reducing the search space and preventing *backtracking*).

Our contribution in this paper, which is an extended version of our workshop paper [12], is to show, using a novel case study from the domain of SPL testing, that metamodelling and Story Driven Modelling (SDM)[3] are suitable techniques for rapid prototyping and supporting maintainability. To this end, we investigate the following two scenarios:

1. An existing, hand-written system (in our case the constraint solver of MoSo-PoLiTe) is regarded as a stable black-box and is only to be optimized. We show that SDMs are well suited for rapid prototyping the flattening (feature model to CSP) transformation, evaluate various optimization strategies to improve the performance of the constraint solver, and present measurement results for the MoSo-PoLiTe SPL testing framework. This was the main focus of our workshop paper [12].
2. A system (in our case the entire MoSo-PoLiTe framework, i.e., constraint solver *and* flattening transformation) is to be either implemented from scratch or completely re-engineered in a model-driven fashion via rapid prototyping using metamodelling and SDMs. To provide a quantitative and qualitative *comparison* of the hand-written system with the re-engineered implementation, we present measurement results for efficiency, memory consumption and implementation effort. Finally, to compare the *maintainability* of both systems, we discuss a real-world change request that we implemented for both systems.

The paper is structured as follows: In Section 2 we introduce our running example and define the necessary concepts used in the rest of the paper. Section 3 discusses the transformation rules that flatten a feature model to a CSP and explains how the CSP solver works. In Section 4, various optimization strategies are investigated and we show how these ideas can be rapid prototyped by translating them almost 1-to-1 in concise graph transformation rules. Corresponding optimization results are presented in Section 5.

For the second scenario, an overview of the re-engineered implementation is provided in Section 6 with a systematic and critical comparison of both systems with respect to runtime efficiency, memory consumption and implementation effort in Section 7. The change request used to evaluate aspects related to the maintainability of both systems is motivated in Section 8, while the corresponding implementation of the change request for the existing and re-engineered system is discussed in Section 9. Section 10 gives an overview of related work and Section 11 concludes the paper.

2. Software Product Line (SPL) case study

An SPL architecture provides a systematic means of deriving different applications from a common architecture family, reusing common *features* (units of functionality) in the process [1]. SPLs are increasing in relevance and importance as various domains strive to cope with the challenges of supporting a high degree of variability. The SPL paradigm, already applied successfully in various application scenarios, promises increased software quality, reduced development and maintenance costs, and a decreased time-to-market [2].

The systematic testing of SPLs, however, is non-trivial as a high degree of variability implies a vast number of possible products. Developing a feasible strategy for testing SPLs, which reuse the same software components in very different combinations and contexts, poses quite a challenge [3], as testing every valid product individually using classical approaches quickly becomes infeasible with respect to time and cost, even for a moderate degree of variability [4].

An established strategy is to determine a representative subset of products which are tested in lieu of the complete product line. Determining an *optimal* subset of products with respect to a chosen test metric is, however, NP-hard as it can be mapped to the minimum cardinality hitting set problem [5]. Many approaches, thus, use suitable heuristics to guide the choice [5–7].

In the rest of this section, basic SPL concepts are introduced together with our running example.

A *feature f* represents a system property that is relevant to some stakeholder [13]. Given the set of all features $F = \{f_1, f_2, \ldots, f_n\}$, a *Product Configuration* $PC \in \mathcal{P}(F)$ is a

[3] A concrete language for specifying programmed graph transformations.