Chapter 15
Reducing Enterprise Product Line Architecture Deployment and Testing Costs via Model Driven Deployment, Configuration, and Testing

Jules White
Virginia Tech, USA

Brian Dougherty
Vanderbilt University, USA

ABSTRACT

Product-line architectures (PLAs) are a paradigm for developing software families by customizing and composing reusable artifacts, rather than handcrafting software from scratch. Extensive testing is required to develop reliable PLAs, which may have scores of valid variants that can be constructed from the architecture's components. It is crucial that each variant be tested thoroughly to assure the quality of these applications on multiple platforms and hardware configurations. It is tedious and error-prone, however, to setup numerous distributed test environments manually and ensure they are deployed and configured correctly. To simplify and automate this process, the authors present a model-driven architecture (MDA) technique that can be used to (1) model a PLA's configuration space, (2) automatically derive configurations to test, and (3) automate the packaging, deployment, and testing of con-figurations. To validate this MDA process, the authors use a distributed constraint optimization system case study to quantify the cost savings of using an MDA approach for the deployment and testing of PLAs.

INTRODUCTION

Emerging trends and challenges. *Product-line architectures (PLAs)* enable the development of a group of software packages that can be retargeted for different requirement sets by leveraging common capabilities, patterns, and architectural styles (Cements 2001). The design of a PLA is typically guided by *scope, commonality, and variability*

DOI: 10.4018/978-1-61692-874-2.ch015

(SCV) analysis (Coplien 1998). SCV captures key characteristics of software product-lines, including their (1) *scope*, which defines the domains and context of the PLA, (2) *commonalities*, which describe the attributes that recur across all members of the family of products, and (3) *variabilities*, which describe the attributes unique to the different members of the family of products.

Although PLAs simplify the development of new applications by reusing existing software components, they require significant testing to ensure that valid variants function properly. Not all variants that obey the compositional rules of PLA function properly, which motivates the need for powerful testing methods and tools. For example, connecting two components with compatible interfaces can produce a non-functional variant due to assumptions made by one component, such as boundary conditions, that do not hold for the component to which it is connected (Weyuker 1998).

The numerous points of variability in PLAs also yield variant configuration spaces with hundreds, thousands, or more possible variants. It is therefore crucial that PLAs undergo intelligent testing of the variant configuration space to reduce the number of configurations that must be tested. A key challenge in performing intelligent testing of the solution space is determining which variants will yield the most valuable testing results, such as performance data.

Solution approach → Model-driven testing and domain analysis of product-line architectures. Model-driven Architectures (MDA) (Karsai 2008, Brown 2008, Paige 2009) are a development paradigm that employs models of critical system functionality, model analysis, and code generation to reduce the cost of implementing complex systems. MDA models capture design information, such as software component response-time, that are not present in third-generation programming languages, such as Java and C++. Capturing these critical design properties in a structured model allows developers to perform analyses, such as queuing analyses of a product-line architecture, to catch design flaws early in the development cycle when they are less costly to correct.

A further benefit of MDA is that code generators and model interpreters can be used to traverse the model and automatically generate portions of the implementation or automate repetitive tasks (Trujillo 2007). For example, Unified Modeling Language (UML) models of a system can be transformed via code generation into class skeletons or marshalling code to persist objects as XML. Model interpreters can be used to automatically execute tests of code using frameworks (Chen 2007), such as Another Neat Tool (ANT) and JUnit.

MDA offers a potential solution to the challenges faced in testing large-scale PLAs. MDA can be used to model the complex configuration rules of a PLA, analyze the models to determine effective test strategies, and then automate test orchestration. Effectively leveraging MDA to improve test planning and execution, however, requires determining precisely what PLA design properties to model, how to analyze the models, and how best to leverage the results of these analyses.

This chapter focuses on techniques and tools for modeling, analyzing, and testing PLAs. First, we introduce the reader to *feature modeling* (Kang 1990, Asikainen 2004, Kang 2002), which is a widely used modeling methodology for capturing PLA variability information. Second, we describe approaches for annotating feature models with probabilistic data obtained from application testing that help predict potentially flawed configurations. Next, we present numerical domain analysis techniques that can be used to help guide the production of PLA test plans. Finally, we present the structure and functionality of a FireAnt, which is an open-source Eclipse plug-in for modeling PLAs, performing PLA domain analysis to derive test plans, and automating and orchestrating PLA testing for Java applications