Taking a Glimpse at Reengineering Challenges in Evolution Towards Dynamic Software Product Lines

Mahdi Derakhshanmanesh

University of Koblenz-Landau,
Institute for Software Technology

manesh@uni-koblenz.de

Abstract

To tame the complexity of engineering customizable software, systems are built as families of products that share a common core. As customers desire the automatic and smart adjusting of their software to various contexts of operation, runtime reconfiguration capabilities need to be added. In this extended abstract, we sketch reengineering challenges to be tackled when evolving classic software product lines into such dynamic software product lines.

1 Introduction

To meet the market's demand for highly customizable software and to achieve systematic reuse of artifacts (e.g., source code, architectural designs, documentation) Software Product Line Engineering (SPLE) [PBvdL05] includes the explicit modeling of variability in terms of *features* – usually in the form of and/or trees, or similar. A *product variant* is described by a *configuration* of features. These are linked to *variations* (e.g., alternate implementations) at *variation points* in the artifacts. Product derivation is performed once: as a part of the build process.

The SPLE approach produces unique product variants. In some application domains, this is not sufficient, e.g., in the entertainment area, users demand more customization possibilities as their preferences change, especially after deployment. Reconfiguration during operation is at the heart of certain products. For instance, in *ambient assisted living*, it is crucial to reconfigure the software that controls the sensors that are observing a patient. In case of a potential cerebrovascular accident (stroke), additional sensors may be enabled, and doors of the apartment may be unlocked for ambulance men.

There are different approaches to develop such context-aware, self-adaptive systems. Assuming that feature models and SPLE processes can guide the development of complex software in a systematic way, there is a trend to leverage related knowledge and techniques by applying them at runtime. This effort is discussed under the umbrella of Dynamic Software Product Lines (DSPLs) [HHPS08] in literature.

Recent works have especially focused on development methodologies for DSPLs. In this extended abstract, we present a preliminary list of challenges that have to be tackled when moving from an existing Software Product Line (SPL) towards a dynamic one.

2 Challenges

We assume that an SPL is given, i.e., variability models as well as a software platform exist, and that user needs or marketing decisions require that the new products must react to environmental changes autonomously and quickly.

In the following, a preliminary list of potential *challenges* (and sub-challenges) is presented in the form of questions. The list is based on the author's experience in the fields of self-adaptive software and the adoption of SPLE in the automotive industry.

C1: What features and variations can be bound when and in which temporal order? In classic SPLE, variability is encoded in terms of *feature models* and the order of choosing features (usually) does not matter, as the product is generated, once a stable configuration is given.

For DSPLs, it is important to analyze which choices of features and depending variations can be made in which order during reconfiguration to meet the new requirements related to dynamism at runtime. Assuming that parts of the product configuration can change at runtime, it is necessary to define further constraints on the feature models, especially regarding *binding time* and the conditions for morphing from one product variant to another.

In terms of related *sub-challenges*, one may attempt to apply configuration changes in groups, instead of applying each change step by step. The *reconfiguration engine* (a.k.a. controller) may propose an optimized, i.e., a prioritized list of changes to apply, and an administrator may check and confirm. Additionally, severe issues will arise if the implemented platform cannot support mechanisms for loading and binding of variations at runtime. These issues must

be detected and prevented early.

Understanding the possible application states in terms of features and the technically possible binding times for variations in the existing artifacts is an essential preparative goal.

C2: How to extend the legacy product line with components for self-adaptation? Most software products are not shipped with embedded controller components. DSPLs require a full infrastructure with sensors, effectors, and controllers to achieve reconfiguration at runtime, though.

In order to make existing software adaptive, it is necessary to identify the needed sensor data and mapping it to appropriate effecting mechanisms. These must be designed and implemented (i.e., integrated into the legacy platform) in a way that does not break the existing architecture.

In terms of related *sub-challenges*, it is non-trivial to locate the right spots for placing software sensors as well as effectors in artifacts of an existing SPL. Given that features can be activated and deactivated, not every sensor and effector will be needed at all times and hence, these dependencies must be detected on appropriate abstractions of the existing SPL platform, too. Furthermore, administrators need to be supported with a control panel to observe and influence the software's changing behavior.

A systematic analysis of the legacy SPL's platform and a transformation into an instrumented extended platform (e.g., guided by *evolution patterns*) is a prerequisite goal when moving towards DSPLs.

C3: How to ensure that the non-adapted products function as before? As the assumed background in this extended abstract is the evolution of an existing SPL towards a DSPL, for sure new requirements trigger this process. Nevertheless, we believe that in most cases, it is desirable that the behavior of existing products shall remain unchanged to ensure backwards compatibility, and to avoid the maintenance of two branches: one for the old SPL and one for the new DSPL.

Introducing the possibility to dynamically change a product configuration at runtime is a challenge, even when the whole system is developed from scratch. Ensuring that the product variants of the legacy SPL maintain their behavior in the DSPL (i.e., in the presence of a controller) is non-trivial. In fact, one may attempt to limit adaptivity with regard to this requirement in order to minimize risks.

Guaranteeing the exact functionality of legacy products (non-functional capabilities may change) is a high-priority goal during forward engineering.

C4: Which additional parts of the infrastructure need to be deployed? For classical SPLs, the deployed product is clearly defined. In the case of DSPLs, a product variant is still scoped, because the major abstraction – i.e., the commonly used feature models – forms a *closed configuration space*.

Customers like to pay only for the features they immediately require. Depending on the target platform's capabilities and capacities, deploying the full DSPL infrastructure and the source code for all product variants may not be a viable option. Nevertheless, fast reconfiguration requires the presence of additional (initially inactive) software artifacts.

Based on the legacy SPL's infrastructure, a given feature configuration, and a set of adaptation rules, the goal is to compute the subset of needed artifacts and their individual setup needed for deployment.

C5: How can we integrate the evolution and self-adaptation loops? According to Lehman's laws, software decays and DSPLs are no exception. Modernization often follows an approach where the legacy system is analyzed, an abstraction is derived such that changes can be applied at an appropriate level of granularity, and finally, the new software is constructed, e.g., using generative techniques.

Runtime reconfiguration in the context of DSPLs aims at providing a means to react to changes in requirements or resources, but this approach is limited to foreseen features. Maintenance (corrective or adaptive) is necessary to face unforeseen situations.

We believe that, in order to successfully move towards an *open configuration space*, the encoding of existing reengineering techniques in the form of adaptation rules is a goal that can be the basis for a more immediate form of software evolution [DEAT11].

3 Summary

In this extended abstract, we scratched only the surface of challenges related to the evolution of existing software product lines towards dynamic software product lines. Nevertheless, we are sure that it can serve as a trigger for fruitful discussions that lead to the extension of this work. Regarding future directions, we need to further analyze the challenges and we believe there is a strong potential in applying reengineering techniques for realizing self-adaptation, too.

References

[DEAT11] M. Derakhshanmanesh, J. Ebert, M. Amoui, and L. Tahvildari. Introducing Adaptivity to Achieve Longevity for Software. In *SE 2011 Workshopband*, volume P-184, pages 59–70. GI, 2011.

[HHPS08] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid. Dynamic Software Product Lines. *Computer*, 41(4):93–95, April 2008.

[PBvdL05] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer, 2005.