A Variability-Centric Approach to Instantiating Core Assets in Product Line Engineering*

Soo Ho Chang, Soo Dong Kim, and Sung Yul Rhew

Department of Computer Science
Soongsil University, Seoul, Korea
shchang@otlab.ssu.ac.kr,
{sdkim, syrhew}@comp.ssu.ac.kr

Abstract. As a key activity in product line engineering (PLE), instantiation is a task to generate target applications by resolving variability embedded in core assets. However, instantiation is often conducted in manual and ad-hoc fashion, largely replying on domain knowledge and experience. Hence, it can easily lead to technical problems in precisely specifying decision model consisting of product-specific variation points and variants, and in handling inter-variant conflicts/dependency. To overcome this difficulty, it is desirable to develop a systematic process which includes a set of systematic activities, detailed instructions, and concrete specification of artifacts. In this paper, we first propose a meta-model of a core asset to specify its key elements. Then, we represent a comprehensive process that defines key instantiation activities, representations of artifacts, and work instructions. With the proposed process, one can instantiate core assets more effectively and systematically.

1 Introduction

Product line engineering (PLE) is one of the recent and effective reuse approaches, and it consists of two processes; *domain engineering* and *application engineering*. Domain engineering is to develop a core asset which captures common functionality and quality attributes among family members in the domain. Application engineering is to generate target applications by instantiating the core asset with a product-specific decision resolution model (DRM)[1].

In PLE, modeling the variability among applications is a key activity because it allows a number of potential applications to reuse the same core asset [2]. A harder part of modeling variability is to identify the conflicts and dependencies among variation points and variants. Once the variability model is constructed, it is further refined and designed into a decision model (DM) which specifies concrete variation points and their relevant variants.

Once a core asset with a DM is constructed as either a model or an implementation, it is instantiated for each application. A key step of instantiation is to define a DRM which specifies application-specific variants for the variation points, and to bind the variants to relevant variation points of the core asset. However, this instantiation is

* This work was supported by grant No. (R01-2005-000-11215-0) from the Basic Research Program of the Korea Science & Engineering Foundation.

J. Münch and M. Vierimaa (Eds.): PROFES 2006, LNCS 4034, pp. 334 – 347, 2006.

carried out largely in manual and ad-hoc fashion, replying on domain knowledge and experience. Moreover, current research works on core asset instantiation have not yet identified the comprehensive instantiation process and detailed instructions.

In this paper, we define a process to instantiate core assets to application specific assets. The process consists of activities which have detailed steps and instructions and produces an instantiated core asset as the final deliverable. We first define a meta-model of core assets, i.e. the key elements and their representation of core assets in section 3. Then, we present an instantiation process and instructions in section 4. Then, we assess our proposed framework with process evaluation criteria in section 5.

2 Related Works

PuLSE (Product Line Software Engineering) is a process for developing product line developed by fraunhofer institute for experimental software engineering (IESE) [3]. It consists of three sub-elements; *Deployment Phases*, *Technical Components*, and *Support Components*. *Deployment phases* is to produce reusable assets and products using related technical components of *PuLSE–BC*, *PuLSE-Eco*, *PuLSE-CDA*, *PuLSE-DSSA*, *PuLSE-I*, and *PuLSE-EM*.

PuLSE-I is used to develop applications and it includes an activity of instantiating product line model and reference architecture [4]. The product line model which is essentially an object model and storyboards representing a core asset is hierarchically resolved with a decision model and application specific characteristics. The reference architecture is refined into an intermediate architecture which embeds an architectural variability and it defines an architectural resolution decision. This identifies activities for instantiating, however, step-wise and detailed instructions are not given. Moreover, elements of each artifact are not defined.

KobrA is a component-based product line engineering method based on UML. It consists of *Framework Engineering* and Application *Engineering*[1]. Framework engineering defines a framework which is a set of reusable assets among family members and Application engineering is to produce applications by instantiating the framework with a decision model. During Instantiation activity, a generic *Komponent* is translated into a specific Komponent by using DM. DM is tailored to a DRM for a specific application. A process to instantiate core assets is also represented by analyzing the overlap between a framework and application requirements and tailoring a Komponent framework. This work defines a process of instantiation at macro level. The instructions given here need to be further refined in order to automate the instructions.

Deelstra's work starts with motivation that deriving products from shared software assets is a time-consuming and expensive activity contrary to the popular belief [5]. This work presents a product derivation framework which defines a number of terminologies, product family classifications according to two dimensions of scope, and generic software derivation process. It also presents a set of identified problems and issues associated with product derivation based on a case study at two large industrial organizations. Case studies conducted in this work show well application of their proposed framework with appearance of well known company, explicit number of components, variation points, LOC, etc.