VULCAN: A Workbench for Feature-Oriented Product Line Software Development

HYESUN LEE[†1] JIN-SEOK YANG[†1] KYO CHUL KANG[†2]

1. Introduction

Software developers, especially embedded system developers, are faced with fierce market competition with: diverse market needs, ever increasing number of features, rapidly changing technologies, and time-to-market pressure. To survive in this environment and enhance their competitiveness in the market, developers are searching for methods and tools to increase their productivity and improve software quality. As a result, software product line engineering (SPLE) methods and support tools have gained popularity ([1, 2]).

There are several methods/tools ([3, 4, 5, 6, 7]) that support SPLE. Existing methods such as [3, 4, 5] provide mechanisms for instantiating products from assets that are created manually. Other methods ([6, 7]) provide languages for describing product line architectures and integration mechanisms, but development of assets need to be done manually. With the existing method ([3, 4, 5, 6, 7]), *the behavior of systems is not visible* thus it is difficult to understand and maintain the systems.

To address this limitation, we have developed a new CASE tool, called VULCAN, that supports the entire feature-oriented product line software development life cycle ([8]). VULCAN uses parameterization of assets with features and instantiation of applications through feature selection as others. However, it also provides architecture models/patterns that make the behavior of systems visible; *user interface, control, and computation components are separated in architecture models*, and *control components can be developed based on controller specification models* provided by VULCAN. Based on these models, VULCAN supports parameterization of product line controller specifications (rather than code asset) that can be instantiated to application controller specifications by selecting feature sets. The application controller specifications can be verified and code can be generated automatically from the verified model. Also, VULCAN can support flexible configuration of various deployment architectures by separating components from components communication mechanisms.

The details of VULCAN are introduced in the following section, and then we conclude this paper in section 3.

2. VULCAN Workbench

We first briefly introduce the overview of the workbench and explain how it supports SPLE processes. Then, each of the tool sets comprising the workbench is explained in detail.

†1 Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH)
†2 Division of IT Convergence Engineering (ITCE), POSTECH

2.1 Overview of VULCAN

VULCAN includes various tools for the entire phases of feature-oriented SPLE, as shown in Figure 1. It supports both proactive and extractive approach to SPLE, and architecture-model-based development of product line software. It contains open software and freeware, and most of the tools are Eclipse plug-in applications. These tools are integrated on the Eclipse platform, thus the workbench can be extended easily by adding new plug-ins.

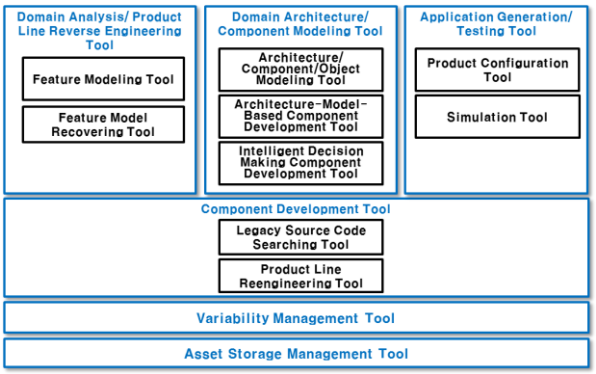

Figure 1. Overview of VULCAN workbench

How the tool sets support the feature-oriented SPLE process is explained in the following subsection.

2.2 Engineering Process Using VULCAN

The feature-oriented product line software development consists of two engineering processes:

- **Domain engineering:** the creation of a feature model and the development of product line assets with embedded variable (i.e., optional/alternative) features
- **Application engineering:** the instantiation of applications from the assets through feature selection

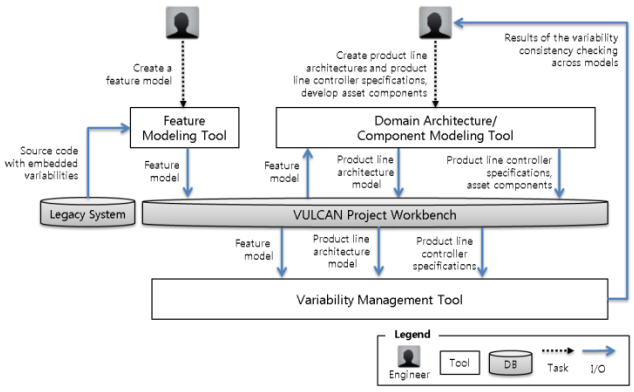

Figure 2. Tools for supporting domain engineering process

Figure 2 depicts how the workbench is used in the domain engineering process. An engineer(s) firstly models common and variable features of the product line with the *feature modeling tool set*. Based on the feature model, the engineer creates product line architecture models, specify product line controllers, and develop asset components, using the *domain architecture/component modeling tool set*. The *variability management tool* checks the consistency of variability across models and shows the results to the engineer.

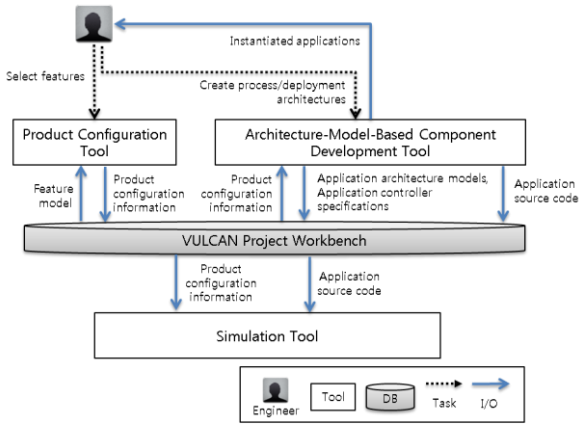

Figure 3. Tools for supporting application engineering process

Figure 3 describes how the application engineering process is supported by the workbench. The engineer first selects variable features of the feature model using the *product configuration tool*. Based on the selection, the *architecture-model-based component development tool set* instantiates application architectures and application controller specifications from the assets. The tool set also automatically verifies the specifications and generates application controllers (code components) from the specifications. The engineer configures process/deployment architectures with various component connection mechanisms using the tool set, and the process code is generated from the architectures. Then, the application code is compiled and linked.

Each of the tool sets comprising VULCAN is explained in detail in the following subsection.

2.3 Tool sets Comprising VULCAN

2.3.1 Domain Analysis/Product Line Reverse Engineering Tool

In the domain engineering process, commonalities and variabilities in a domain is analyzed in terms of features and modeled in the feature model ([9]). The *feature modeling tool* supports this activity. Using the tool, domain engineer(s) can easily add/modify/delete features (of which some may be *mandatory*, *optional*, or *alternative*) and feature dependencies (*require* or *exclude*). Also, for each feature, its binding time, information source, description, and attributes can be defined. The tool validates a feature model and notifies the engineer if any inconsistency happens (e.g., a feature and its parent have *exclude* dependency).

For an extractive approach to SPLE, the *feature modeling recovering tool* supports product line reverse engineering, i.e., analyzing variation points embedded in source code and recovering a feature model from them semi-automatically. The details of this tool can be found in [10].

2.3.2 Domain Architecture/Component Modeling Tool

In the domain engineering process, based on a feature model (created using the *domain analysis/product line reverse engineering tool set* in subsection 2.3.1), product line architecture/components are modeled using the *domain architecture/component modeling tool set*. As we mentioned in section 1, VULCAN supports architecture- model-based product line software development. We will focus on the architecture modeling tool and architecture-model-based component development tool in this subsection.

Using the *architecture modeling tool*, engineers can create product line architecture models while embedding variable features into the models. The product line software developed with VULCAN is based on the common architecture model as shown in Figure 4. As explained in section 1, to make the system behavior visible, we separate user interface, control, computation, and data management components ([11]). The architecture model is a "logical architecture"; components of the architecture will be allocated to processes/nodes later.

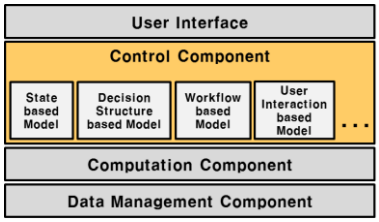

Figure 4. Underlying architecture model

A controller integrates components and interacts with other controllers, and controller design embodies architectural design decisions. Therefore, we focus on the development of controllers using various controller specification models; we will not address development of other components in this paper.

By analyzing industrial software products of various domains ranging from mobile game software to factory management applications, we could identify several controller specification models that are commonly used. Each of models emerges based on the characteristics of an application domain. For instance, mobile game software is user-interaction-scenario-based while factory control applications are typically state-based (i.e., they react to external events based on the state of the factory). Software that controls a system based on the states of the system, for example, can be developed using the state-based controller specification models ([12]). By applying an appropriate model to develop product line controllers, we can easily understand and maintain the behavior of controllers and thus achieve high productivity and high quality of software as a result. We currently provide tools that support the following four kinds of controller specification models: state-based model ([12]), decision-structure-based model, workflow-based model, and interaction-scenario-based model.

To develop product line controllers, using the *architecture-model-based component development tool*, the engineer first selects an appropriate controller specification model based on the characteristics of an application domain. Then, the engineer develops the specification with embedded variable features as variation points while integrating reusable computation components with the specification.

To meet the variability requirements of a product line, we must embed variation points into the specifications. We use a parameterization approach because we only have to maintain parameterized specification (we do not have to maintain application-specific instances). Variable features defined in a feature model are used to parameterize the specifications. When product/design decisions are made by selecting features during application engineering, these specifications are instantiated to application controller specifications ([8]).

In the application engineering process, using the *architecture-model-based component development tool*, the engineer can instantiate an application from assets, based on product configuration information (This information is created using the *product configuration tool* in subsection 2.3.3.). It consists of the following five activities.

1) Instantiation of an application architecture model: The engineer can create an application architecture model instance from a product line architecture model by selecting appropriate features. In the product line architecture model, components and relationships between components mapped to the selected variable features are included in the application architecture model; while elements mapped to unselected features are excluded.

2) Instantiation of application controller specifications: The application controller specification instantiator of the underlying controller specification model automatically creates application controller specifications from product line controller specifications based on the selected features.

3) Verification of the specifications and code generation: The specification instance can be verified using the corresponding specification verification tool of the controller specification model. After the specification is verified, then Java source code for the controller(s) can be generated from the specification using the application source code generation tool. The verification and generation tools are based on our earlier tools ([2] [15]) which are integrated in VULCAN.

4) Specification of process and deployment architecture: The components, including controllers, created in the domain engineering process are "logically connected." We need deploy them to process and deployment architectures (See Figure 5) and also select component connection mechanisms (See Table 1.) to physically bind these components. The process architecture ([16]) is used to define concurrent processes, each of which has its own thread of control and may be allocated to any node on the network. One or more controllers may be allocated to each process. The engineer can select a connection mechanism for each interaction relationship between components. The connection mechanisms that the engineer can select are in Table 1 ([17]). After a process architecture is created, the engineer develops a deployment architecture by allocating each process to a network node using the application deployment configuration tool. The engineer can define the network information (e.g., IP address and port number) for each node to finalize the deployment.

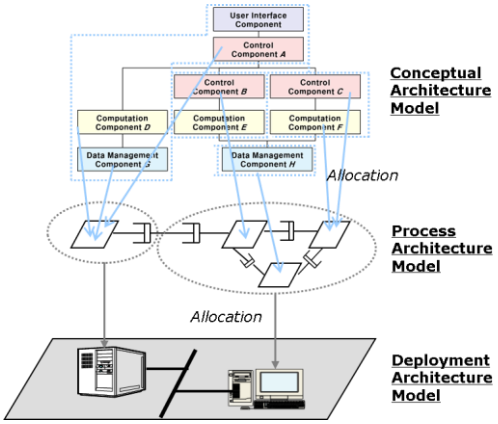

Figure 5. Deployment configuration relationship

Table 1. Component Connection Mechanisms ([17])

Subsystem Level	Data Flow						Control Flow	
Process Level	Message Queue		Message/Reply		Shared Data		Event	
Programming Interface Level	Socket		Procedure Call		Shared Memory		Implicit Invocation	
Implementation Technique Level	Local Socket	Java Socket	Local Procedure Call	Remote Procedure Call (RMI)	Local Shared Memory	Remote Shared Memory	Local Observer Pattern	Remote Observer Pattern

5) compiling/linking of code: The source code is compiled and linked with component binding mechanisms for components and processes defined in the previous activity.

2.3.3 Application Generation/Testing Tool

In the application engineering process, the engineer can select variable features of a feature model using *product configuration tool.* The tool verifies the feature selection and notifies the engineer when any invalid selection is made (e.g., selecting features that are mutually exclusive). The instantiation process based on the product configuration was depicted in subsection 2.3.2.

After the application is created, it can be tested based on simulation using the *simulation tool*. This tool provides a method for modeling a virtual environment where control software (the application) will be embedded and tested ([2]). Details of the tool can be found in [18].

2.3.4 Component Development Tool

This tool set supports development of components based on legacy code components in the context of the extractive approach to SPLE. In the extractive approach, some legacy components may be reused without modification, but to satisfy the required quality attributes and/or the functional requirements, other components might need to be modified. The *legacy source code searching tool* supports identifying similar Java code by searching legacy code database ([19]). Using the *product line reengineering tool*, the engineer can analyze the legacy system structures and separate reusable asset components from the

legacy systems, based on “component types” ([20]).

2.3.5 Variability Management Tool

Consistency across different lifecycle artifacts is an important issue in software engineering. In SPLE, validating consistency becomes even more complicated because product line assets have embedded variabilities. This tool supports validating consistency of mandatory/optional/alternative features across product line assets (requirements, architectures, and components). The details of the validation method can be found in [10].

3. Concluding Remarks

The proposed workbench, VULCAN, is composed of various tools for supporting the entire phases of feature-oriented SPLE and especially provides tools for supporting architecture-model-based product line software development. The workbench provides the following benefits to software engineers:

- Maintainability of asset components can be increased by separating components that frequently change (e.g., user interface, controller) from others.
- With a specification-based automatic verification and code generation approach to developing controllers, we can acquire controllers that are more error-free than code-based approach. Also, we can easily change the specification and generate new controllers.
- Controllers can be separated into global and local controllers, and a distributed control network can easily be configured by defining/changing the deployment architecture.
- Components that rarely change (e.g., computation components) can be tested and adapted/reused thus achieving high adaptability/reusability.

Our tool has been applied to various product lines including glucose monitoring systems and elevator control systems. We could experience that maintainability of the assets has improved substantially because a large portion of the assets are specifications rather than code and it is easy to change the specifications and generate application code.

VULCAN is now being applied to continuous casting process computers at a steel works. We will improve VULCAN based on their feedback and also will make sure that it can scale up to support large projects. We also plan to identify and define other types of architecture patterns to support software development in various application domains.

Reference

1) Clements, P. and Northrop, L: Software product lines: practices and patterns. Addison-Wesley Professional (2001)
2) Kim, K. et al: ASADAL: a tool system for co-development of software and test environment based on product line engineering. In: 28th International Conference on Software Engineering, 783-786 (2006)
3) GEARS, BigLever Software, Inc. Austin, TX, USA: www.biglever.com.
4) Pure::Variants, Pure-systems GmbH. Magdeburg, Germany: www.pure-systems.com.
5) Dhungana, D. et al: DOPLER: an adaptable tool suite for product line engineering. In: 11th International Software Product Line Conference, Second Vol., 151-152 (2007)
6) Dashofy, E.M. et al: A highly-extensible, XML-based architecture description language. In: Working IEEE/IFIP Conference on Software Architecture, 103-112 (2001)
7) van Ommering, R. et al: The Koala component model for product families in Consumer Electronics Software. IEEE Computer. 33, 2, 78-85 (Mar. 2000)
8) Kang, K.C. et al.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering, 5, 143–168 (1998)
9) Kang, K.C. et al.: Feature-oriented domain analysis (FODA) feasibility study. Technical report, CMU/SEI-90-TR-21, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University (Nov. 1990)
10) Le, D.M. et al.: Validating Consistency between a Feature Model and its Implementation (2012) (Unpublished paper)
11) Lee, H. et al.: VULCAN: Architecture-Model-Based Software Development Workbench. In: Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (2012)
12) Harel, D. 1998. On visual formalisms. Communications of the ACM. 31, 5 (May. 1998), 514-530.
13) Statechart. IEICE Trans. Inf. & Syst. E82-D, 2 (Feb. 1999), 398-411.
14) Lee, H., et al: Experience report on using a domain model-based extractive approach to software product line asset development. In: 11th International Conference on Software Reuse, 137-149 (2009)
15) Ko, K.I. and Kang, K.C.:ASADAL/PROVER: A toolset for verifying temporal properties of real-time system specifications in Statechart. IEICE Trans. Inf. & Syst. E82-D, 2, 398-411. (Feb. 1999)
16) Gomaa, H: A software design method for real-time systems. Communications of the ACM. 27, 9 (Sep. 1984), 938-949 (1984)
17) Kim, K: Design and implementation of layered connectors for software component composition. Master thesis, Dept. of CSE, POSTECH (1998)
18) Lee, J. et al.: A real world object modeling method for creating simulation environment of real-time systems. In: Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 93-103 (2000)
19) Kim, J. et al.: Towards intelligent code search engine. In: 24th AAAI Conference on Artificial Intelligence, pp. 1358-1363 (2010)
20) Cho, S.: Asset component identification and re-engineering method for an extractive software product line engineering. Master thesis, Dept. of ITCE, POSTECH (2012)

Acknowledgments This research was supported by the National IT Industry Promotion Agency (NIPA) under the program of Software Engineering Technologies Development; by World Class University program funded by the Ministry of Education, Science and Technology through the National Research Foundation of Korea (R31-10100); by the Agency for Defense Development (ADD), Korea, and has been performed in conjunction with REALTIMEVISUAL Co., Korea.