Coordinating Multi-Team Variability Modeling in Product Line Engineering

Deepak Dhungana Rick Rabiser Paul Grünbacher
Christian Doppler Laboratory for Automated Software Engineering
Johannes Kepler University, 4040 Linz, Austria
{dhungana, rabiser}@ase.jku.at paul.gruenbacher@jku.at

Abstract

In product line engineering variability models capture the commonalities and variability of core assets and guide product derivation. In large-scale systems the knowledge that is required for creating and evolving variability models is typically distributed among different heterogeneous stakeholders. For example, sales people usually think in terms of features and monetary resources while developers emphasize architectural elements, software resources, and configuration parameters. This paper is based on experiences from an ongoing industrial research project and proposes an approach for sharing variability knowledge in a multi-team development organization. Our approach allows different teams to create a variability model from their point of view (e.g., for a subsystem they are responsible for). Subsequently all created models are combined to one integrated variability model.

1. Introduction

It has been demonstrated that product line engineering (PLE) can reduce cost and increase productivity and quality through consequent reuse of core assets [8]. Variability management is a key concept in PLE to express the commonalities and variability of core assets and to understand dependencies among them [4]. Variability models can also be seen as building plans for product instantiation and configuration.

Creating a variability model is not trivial as it relies on information spread across the minds of numerous heterogeneous stakeholders. Today's software systems are large and often different development teams are in charge for different parts of the system. The dependencies between the communication structure of a development team and the technical structure of a system have been addressed by Conway's law [1, 5]. Working with large-scale variability models requires mechanisms that support the cooperation of different teams building and evolving such models for the parts of the system they are dealing with. We are facing these challenges in an ongoing research project with Siemens VAI, the world leader in engineering and building plants for the iron, steel, and aluminum industries. In this paper we propose an approach for distributed editing and integration of variability knowledge.

2. Capturing Variability Knowledge

As part of our research in bridging the gap between stakeholders in PLE, we have been developing an integrated variability model [3]. Our model covers product line assets (e.g., components, resources, features) and decisions as shown in Figure 1.

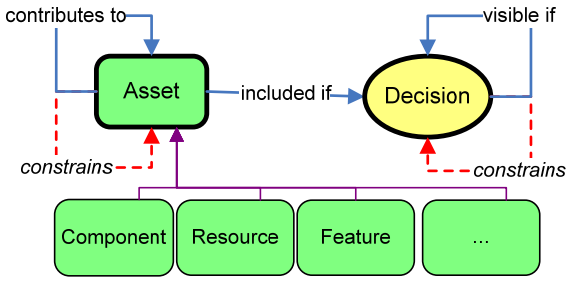

Figure 1: Integrated variability model

Assets address different levels of variability including customer visible properties expressed as features, architectural elements such as components, and implementation level details such as properties. Assets can have structural (*contributes to*) and logical (*constrains*) dependencies. Decisions are used to link the various model elements. A decision can be seen as a variation point, where the user is given an option. The decision taken by a user influences the selection of assets. Decisions can also have dependencies. V*isible if* models the situation that the selection of a certain value makes another decision relevant. Decision can also constrain other decisions.

While this model works well for capturing different PLE aspects such as customer-oriented perspectives

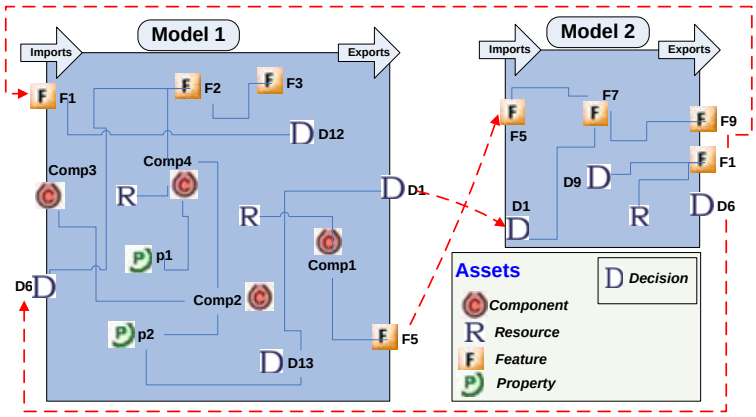

Figure 2: Sharing and merging of variability models

and technical perspectives, it is unrealistic to assume that such a model can be created and evolved by an individual or by a small team for a real-world system. Interaction with our industry partner confirmed that teams typically have detailed knowledge of a small set of subsystems and only rough knowledge about other subsystems [2]. It is therefore important to support the interaction of different teams via distributed variability models linked through model connectors [7] to allow teams to separately create and evolve models. Consistency is ensured through the defined model connectors which allow combining all individual variability models into one integrated variability model.

3. Sharing and Merging Variability Models

Our approach is based on concepts in architecture description languages (ADLs) [6]. Such languages support modeling of large-scale systems by defining interfaces of subsystems and the interaction of subsystems. While ADLs focus on architectural elements such as components or connectors we extend this idea to all elements of our integrated variability model. A team working on a variability model can specify elements of the model (see Figure 1) as public to "export" them. Variability models of other subsystems can then "import" the public elements as part of their own variability model (e.g., when specifying constraints between subsystems). Private elements are internal to a subsystem with no relationships to elements in other models. Distributed teams building individual models only have to know what other elements have an effect on the elements they create. It is not necessary (and often not possible) to know what effect a newly added element has on elements in other models. When introducing a new component to a model, it is easier to describe which other components and resources are needed by this component, rather than modeling where this new component can be used.

The mapping between exported elements of one model and imported elements of another model can be automated. A match can be found for elements of the same type and with the same name. As illustrated in Figure 2, *model 1* exports a decision *d1* and a feature *f5*, which are imported by *model 2*. By merging the individual models a complete variability model is created that guides the instantiation of products in PLE. We have been developing an initial prototype of this capability as part of our ongoing tool development.

References

[1] M.E. Conway, "How Do Committees invent?", *Datamation*, 14 (4), 1968, pp. 28–31.

[2] D. Dhungana, R. Rabiser, P. Grünbacher, H. Prähofer, C. Federspiel, and K. Lehner, „Architectural Knowledge in Product Line Engineering: An Industrial Case Study", Proc. *32nd Euromicro Conf. on Software Engineering and Advanced Applications*, IEEE CS, 2006.

[3] D. Dhungana, "Integrated Variability Modeling of Features and Architecture in Software Product Line Engineering", *Doctoral Symposium, 21st IEEE/ACM Int. Conf. on Automated Software Engineering*, Tokyo, Japan, 2006.

[4] J. Estublier and G. Vega, "Reuse and Variability in Large Software Applications", *Proc. 10th European Software Engineering Conference*, Portugal, 2005, pp. 316–325.

[5] J.D. Herbsleb and R.E. Grinter, "Architectures, Coordination, and Distance: Conway's Law and Beyond". *IEEE Software*, 16 (5), 1999, pp. 63–70.

[6] N. Medvidovic and R.N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", *IEEE TSE*, 26 (1), 2000, pp. 70–93.

[7] N. Medvidovic, P. Grünbacher, A.F. Egyed, and B.W. Boehm, "Bridging Models across the Software Lifecycle", *Journal of Systems and Software*, 68 (3), 2003, pp. 199–215.

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