

Demonstration: Variant and Variability Management with **pure::variants**

Danilo Beuche

pure-systems GmbH, Agnetenstr. 14
D-39106 Magdeburg, Germany
<http://www.pure-systems.com>

Abstract The demonstration will show a round-trip through the development and deployment of product lines with the pure::variants Eclipse plug-in using concrete examples from the embedded software domain. pure::variants is one of the few commercially available tools specifically designed for product line development.

The tool covers all steps of product line development from requirements and variability analysis to product generation. Extended feature models are used for problem domain modeling. Family models are used to represent the variable architecture of product line solution domains independent of the programming or modeling languages used for product line implementation.

1 Introduction

Several important issues have to be considered for tool chains supporting the complete process of variability management:

- Easy, but universal model(s) for expressing variabilities and commonalities should be supported.
- Variability at all levels must be manageable.
- Introduction of new variability expression techniques should be possible and easy.

The pure::variants tool chain outlined in the next sections and presented in the demonstration was developed to meet all these requirements. It is the commercial successor of the research prototype presented in [1].

2 Integration of pure::variants into PLD processes

The pure::variants-based tools are used in different phases of the software development process. The development of product lines is basically divided into two steps. In the first step the problem and solution domains are analyzed, common assets are identified and realized (domain engineering). In the second step the individual products are derived from the product line (application engineering).

Several model types are used to capture the information required to manage variability and variants on the different levels of domain knowledge, software design and implementation. *Feature models* play a key role in this. They allow a uniform representation of variabilities and commonalities of the products of the entire product line. Compared to original works on feature models, pure::variants supports an extended version of this concept. Implementations of the product line are described by family models. They enable the mapping of the problem space to the different implementations. This model type was developed especially for the pure::variants technology, since existing modeling techniques such as UML or SDL were not suitable for this purpose. The variant description model is used to describe a individual product. It describes the product's features and values associated with those features, and it is used to derive the final product from the family models.

2.1 Domain Engineering

The principle operational sequence of such a development process is described in the following, beginning with the identification of the common assets:

1. Analysis of the problems:

Based on a content-wise problem analysis, feature models [2] are build to capture the dependencies between the individual features of the product line. Feature models are easily visualized and conceived.

Based on the experience with the practical use of feature models, the expressiveness of pure::variants's feature models was substantially extended. The support of model hierarchies in particular enables different representations of the problems depending on the user (customer, developer, sales, ...).

2. Design of the solutions:

Starting from the feature model and in combination with further requirements for the software systems, the design of the software solution is performed. The elements of the software solution with their relations, restrictions and requirements are integrated into the family model and hence are available for automatic processing.

The family model is divided into several levels. The highest level is formed by the so-called components. Each component represents one or more functional features of the solutions and consists of logical parts of the software (classes, objects, functions, variables, documentation). In the next level, the physical elements of the solution are assigned to the logical elements. The physical elements can be files that already exist as well as files that are to be created and actions that are to be performed based on the variant knowledge.

The pure::variants technology captures the problems (feature model) and the solutions (family model) separately and independently. This enables a simplified re-use of the solutions and of the feature models in new projects.

3. Realization of the solutions:

The solutions are realized by employing the capabilities of the selected programming language and tools, and by using the additional possibilities of generating variants provided by the pure::variants transformation modules, .

2.2 Application Engineering

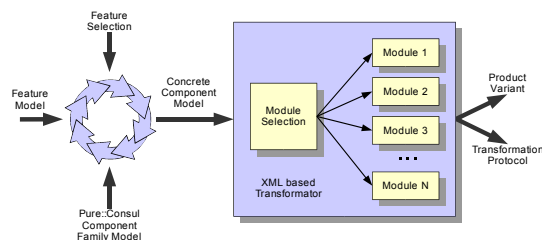


Figure1. pure::variants transformation process

Figure 1 gives an overview of the basic process of creating variants with pure::variants. Once the different models are produced, the remaining steps are performed automatically. The developers of the product line are responsible for providing the feature model as well as the solution description with family models. The user then selects the features. Here, the user can be both a human or a tool that determines the necessary features automatically based on the application. Further processing is performed in two steps. At first pure::variants analyzes the different models. The result is a construction plan from which the customized component of the final product is derived in a second step, called transformation. The transformation step is configurable by the product line developers/users.

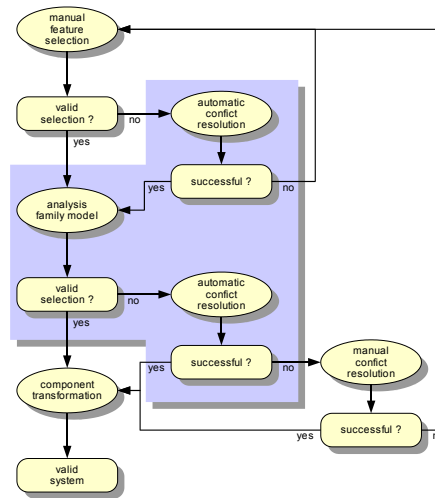


Figure2. Employing pure::variants to create a problem solution

Figure 2 illustrates the tool supported evaluation of feature models with pure::variants in more detail. The main steps of that process are as follows:

1. Determining a valid combination of features:

The user (customer, sales, application developer) selects the features relevant to the problem solution from the feature model. pure::variants checks whether the feature selection is valid and, if necessary, resolves dependency conflicts automatically. This ensures that even complex dependency structures can be efficiently transformed into a valid system.

The result is a combination of features that describes the problem to solve as intended by the developers of the models.

2. Selecting a suitable solution:

Based on the selected features, the family model is used to find a suitable solution. Using the information contained in the family model, the feature selection is analyzed by each component and its logical and physical elements. For each part it is decided whether and in which form it belongs to the solution. Problems that may arise due to further dependencies are resolved automatically if possible or handed over to the user for manual solution (example in figure 3). The strategy for the selection of the best suitable solution is in general manually realized by the users. However, it is possible to integrate problem-specific strategies into pure::variants if necessary, depending on user demands. Thus it is possible to select a solution according to optimization parameters,

e.g. according to the customer's cost model for parts of the component.

The result is a description of the selected solution in form of a component description.

3. Creating the solution:

The customized software solution is created by a transformation process controlled by the component description. During this process all necessary transformation modules are activated and perform the conversions specified in the component description.

Not only the creation of new product lines, but the integration of existing software (re-engineering) into a product line based development in particular is easily possible with pure::variants. For this purpose the development process does not begin with the identification of the common assets but with the (partial) automatic production of the family model based on the already existing software and with the step-by-step construction of the associated feature model by the users. The remaining steps are similar to the process outlined above.

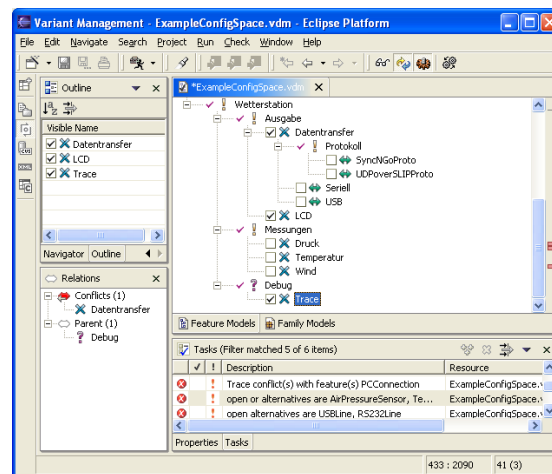


Figure3. Configuration of a weather station product line with pure::variants

3 Conclusion

A promising solution to increase development productivity and quality of software are product lines that support software development not only for one product but for a class of products. Because of the initial identification of common assets, synergies can be exploited that do not come to fruition in classical software design technologies. For a successful use of software product lines, tools are necessary that support the entire process beginning with the design up to the deployment. The efficient management and realization of building variants within the common assets represents a technological challenge. The pure-systems GmbH provides support for this process with its variants management tools based on the pure::variants technology.

References

1. Danilo Beuche, Holger Papajewski, and Wolfgang Schröder-Preikschat. Variability Management with Feature Models. In *Proceedings of the Software Variability Management Workshop*, pages 72–83, University of Groningen, The Netherlands, February 2003. Technical Report IWI 2003-7-01, Research Institute of Mathematics and Computing Science.
2. K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, November 1990.