# Integrating Model-Driven Development and Software Product Line Engineering

Iris Groher[1], Holger Papajewski[2], Markus Voelter[3]
*Siemens AG, CT SE 2, Munich Germany*
*pure::systems, Magdeburg, Germany*
*Independent Consultant, Goeppingen, Germany*
*iris.groher.ext@siemens.com, holger.papajewski@pure-systems.com, voelter@acm.org*

## Abstract

*Software product line engineering aims to reduce development time, effort, cost, and complexity by taking advantage of the commonality within a portfolio of similar products. The effectiveness of a software product line approach directly depends on how well feature variability within the portfolio is implemented and managed throughout the development lifecycle, from early analysis through maintenance and evolution. We present an approach that facilitates variability implementation, management, and tracing by integrating model-driven development and software product line engineering. Appropriate tooling is the basis for a successful application of the new concepts. This paper illustrates how pure::variants and openArchitectureWare are integrated to enable efficient model-driven variant management.*

## 1 Introduction and Motivation

The effectiveness of a software product line approach directly depends on how well feature variability within the portfolio is managed from early analysis to implementation and through maintenance and evolution [1][2]. Commonalities, as well as the flexibility to adapt to different product requirements are captured in core assets. Those reusable assets are created during domain engineering. During application engineering, products are either automatically or manually assembled, using the assets created during the domain engineering process and completed with product-specific artifacts. Products usually differ by the set of features they include in order to fulfill customer requirements. A feature is an increment in functionality provided by one or more members of a product line [3].

Variability management is the activity concerned with identifying, designing, implementing, and tracing flexibility in software product lines (SPLs). Variability of features often has widespread impact on multiple artifacts in multiple lifecycle stages, making it a predominant engineering challenge in software product line engineering (SPLE).

Our approach integrates model-driven software development [4] (MDSD) and product line engineering by providing means for expressing variability on model level. We argue that due to the fact that models are more abstract and hence less detailed than code, variability on model level is inherently less scattered and therefore simpler to manage.

While related publications have focused on describing the approach in general [5], the expression of variability on model level [6] as well as transformation and code generation level [7], this paper describes the integration of the necessary tools. The tool chain is based on Eclipse and integrates pure::variants [8] as a variant management tool and openArchitectureWare (oAW) [9] as a MDSDt toolkit.

The integration of pure::variants and oAW supports seamless model-driven software product line development. Within the oAW tooling, configuration models can be queried and hence, activities can be performed based on the presence or absence of features in a variant model.

The remainder of the paper is organized as follows: Section 2 provides a short introduction to the integrated tools, namely pure::variants and oAW. Section 3 illustrates their integration. Related work is discussed in Section 4. Section 5 summarizes the paper and provides an outlook on future work.

## 2 Introduction to the Tooling

### 2.1 pure::variants

Related products frequently share much of the same software, with only a few differences realizing product-specific functionality. However, much of the challenge of developing related products comes from managing these differences. Variant management addresses this problem by enabling the development of a group of related products as a whole, rather than as individual, independent projects.

pure::variants is a purpose-built variant management tool. It manages product lines and assists in producing individual product variants. Variant management is required in all stages of Product Line development. pure::variants provides an infrastructure for variability modeling and variant definition in all phases of a system development. This allows existing tools to be

augmented to handle variability and variants more efficiently. With its open interfaces, variant information can be used consistently in requirements engineering, during systems design and implementation and also in testing.

**Separates problem and implementation models**

The basic idea of pure::variants is to realize product lines by family models and feature models. Contrary to previous feature model based technologies, pure::variants captures the problem with feature models
and the solutions with family models separately and independently. A family model consists of so-called components. Components represent at least one attribute of the software solution and contain the logical parts like classes, objects, functions, variables and documentation. A feature model specifies the interdependencies of the features of a product line. They allow a uniform representation of all variabilities and commonalities of the products of the product line. pure::variants also supports the use of multiple feature models that are hierarchically linked.

**Transforms requirements into solutions**

During the problem analysis and solution design stages of software development, pure::variants records and administers all models and information, thus assists in the creation of software families implementing product lines. pure::variants can even be used to automatically generate custom-made solutions: A user (developer or customer) selects features required for the desired product from the feature models. pure::variants checks the validity of this selection and if necessary automatically resolves dependency conflicts. A valid selection triggers an evaluation of the family models that contain component definitions consisting of logical and physical parts. The evaluation process results in an abstract description of the selected solution in terms of components. This description controls the transformation process that in creates the custom-made software solutions.

## 2.2 openArchitectureWare

oAW is a framework that integrates a number of tool components for all aspects in model-driven software development (MDSD). oAW is basically a "tool for building MDSD tools". It supports arbitrary import model formats, meta models, and output code formats. At the core there is a workflow engine allowing the definition of transformation workflows as well as a number of prebuilt workflow components that can be used for reading and instantiating models, checking them for constraint violations, transforming them into other models and then finally, for generating code.

Workflow files are XML files that describe the steps that need to be executed in a generator run. Each of these steps is specified with what is called a workflow component. A typical oAW workflow consists of loading one or more models, checking constraints on them, transforming them into other models and then generating code from them.

With a suitable instantiator, oAW can read any model. There is built-in support for EMF [10], various UML tools, textual models, XML, and Visio. Code generation in oAW is done using a language called Xpand. It is an object-oriented template language that supports template polymorphism and template aspects [11]. For checking model constraints, oAW offers a declarative constraints checking language called Check that is very much like OCL.

Model-to-Model transformations can be implemented using a language called Xtend. It is a textual and (more or less) functional language for querying and navigating existing models as well as building new models. The expression sub-language is a simplified version of OCL.

The expression of variability in structural models can be done using tools called XWeave and XVar [6]. They support the implementation of positive as well as negative variability on model and meta model level.

# 3 Integrating pure::variants and oAW

The integration of pure::variants and oAW supports seamless model-driven software product line development. Within the oAW tooling, global configuration models can be queried and hence, MDSD activities can be performed based on the selection of features. Workflow components, model-transformations, and code generation templates can be connected to features. Their execution thus depends on the presence or absence of the feature in the configuration model.

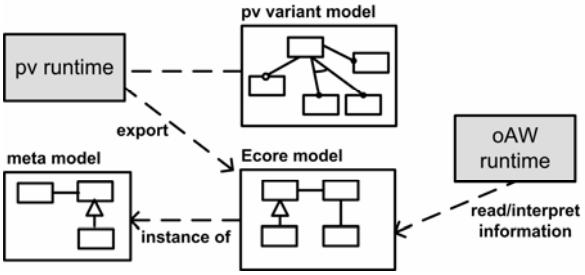


**Figure 1: Integrating pure::variants and oAW**

The data transfer between pure::variants and oAW is done using Ecore [10]. An automatic variant model-export into an Ecore model is provided by pure::variants. The oAW runtime reads this model and interprets the variant information. In order for this integration to work, a meta model for variant models has

been defined. Figure 1 illustrates this integration and the respective models.

Figure 2 shows how either workflow steps or model transformations and code generation templates can be connected to features defined in a global configuration model. Therefore, their execution can depend on the selection of features in the configuration models.

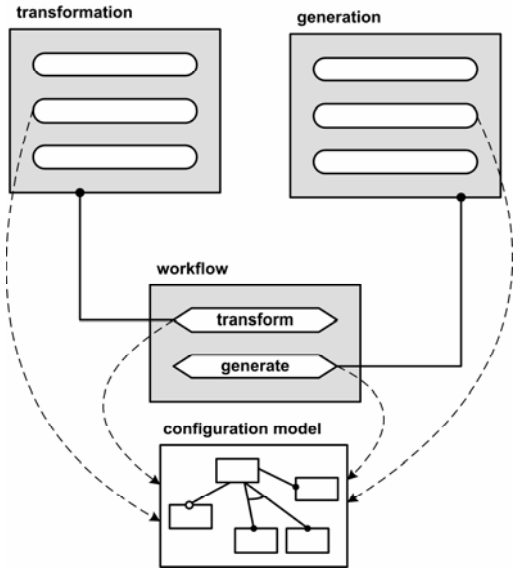


**Figure 2: Linking model transformation and code generation to configuration models**

**Connecting workflow steps with the configuration**

Due to the integration of pure::variants and oAW it is possible to express dependencies between workflow steps and features. Workflow components are then only executed if the respective feature is selected in the configuration model. This is expressed by a surrounding <*feature…*> tag.

For example, we only want to have a logging interceptor installed in a system if the feature *logging* is selected in the global configuration model. The following code snippet shows how this dependency is expressed in a workflow description:

```
<feature exists="logging">
  <component adviceTarget="xtendComponent.ps2cbd"
             class="oaw.xtend.XtendAdvice">
    <extensionAdvices value="logging"/>
  </component>
</feature>
```

**Querying the feature model directly**

We can also access the configuration model directly from within transformations or code generation templates. For example, the following piece of transformation code optionally adds burglar detection facilities to a building. The same function can be called from inside a template (typically, as part of an *IF* statement). The optional transformation steps are only executed in case the respective features are part of the configuration.

```
create System transformPs2Cbd( Building building ):
  ...
  hasFeature("burglarAlarm") ? ( handleBurglarAlarm() -> this) : this;

handleBurglarAlarm( System this ):
  let conf = createBurglarConfig(): (
    configurations.add( conf ) ->
    ...
    conf.connectors.add( connectSimToPanel( createSimulatorInstance(),
                         createControlPanelInstance() ) ) ->
    hasFeature( "siren" ) ? conf.addAlarmDevice("AlarmSiren") : null ->
    hasFeature( "bell" ) ? conf.addAlarmDevice("AlarmBell") : null ->
    hasFeature( "light" ) ? conf.addAlarmDevice("AlarmLight") : null
);
```

**Feature Attributes**

It is also possible to address properties or attributes of features. For example, you might want to be able to configure the volume of the siren in the configuration model. The transformation would read this value from the configuration model and parameterize the siren component instance accordingly. Here's the code:

```
handleBurglarAlarm( System this ):
  ...
  isFeatureSelected( "siren" ) ? (
    let siren = conf.addAlarmDevice("AlarmSiren"):
      siren.configParamValues.add( siren.createParamForLevel() )
    ) : null ->
  ...
);

private create ConfigParameterValue
      createParamForLevel( ComponentInstance instance ):
  setName( "level" ) ->
  setValue((String)getFeatureAttributeValue( "siren", "level" ));
```

## 4 Summary and Future Work

In this paper we have shown how the integration of pure::variants and oAW combines the power of model-driven software development with variability management. This enables more flexible MSDS solutions where users are able to configure generated systems easily. pure::variants is responsible for providing valid configuration information to the MDSD tooling. oAW performs well defined transformation and generation steps based on the presence or absence of available features. The user can customize the transformation and generation steps simply by selecting features from the provided variability model. It is possible to query the variant model from within workflow files, model transformations, and code generation templates.

Due to the open and extensible nature of Eclipse, a seamless integration of pure::variants and oAW is possible. The tooling serves as the basis for a successful application of the aspect-oriented model-driven soft-

ware product line engineering methodology that we envision [5].

In the future we will work on a better visualization of the dependencies between MDSD artifacts and features. For example, the workflow components, transformation steps, and templates that depend on a feature can be highlighted in the same color. This facilitates variant tracing and maintainability of features.

## 5 Acknowledgments

This work is supported by AMPLE Grant IST-033710.

## 6 References

[1] K. Pohl, G. Böckle, and F. v. d. Linden, *Software Product Line Engineering Foundations, Principles, and Techniques.* Berlin: Springer, 2005.

[2] P. Clements, and L. M. Northrop, *Software Product Lines: Practices and Patterns.* Addison-Wesley, 2001.

[3] P. Zave, "FAQ Sheet on Feature Interaction": http://www.research.att.com/~pamela/faq.html

[4] T. Stahl, and M. Voelter, *Model-Driven Software Development:* Wiley & Sons, 2006.

[5] M. Voelter, and I. Groher, "Product Line Implementation using Aspect-Oriented and Model-Driven Software Development", *In Proceedings of the 11th International Software Product Line Conference (SPLC)*, Kyoto, Japan, 2007.

[6] I. Groher, and M. Voelter, "Expressing Feature-Based Variability in Structural Models", *In Proceedings of the Workshop on Managing Variability for Software Product Lines*, Kyoto, Japan, 2007.

[7] M. Voelter, and I. Groher, "Handling Variability in Model Transformations and Generators", *Submitted for publication*, 2007.

[8] pure::variants Variant Management Tool website, http://www.pure-systems.com/3.0.html

[9] openArchitectureWare (oAW) website, http://www.eclipse.org/gmt/oaw/

[10] Eclipse Modeling Framework (EMF) website, http://eclipse.org/emf

[11] AOSD website, http://www.aosd.net