Variability and Evolution in Model-based Engineering of Embedded Systems

Goetz Botterweck
Lero – The Irish Software Engineering Research Centre
Limerick, Ireland
goetz.botterweck@lero.ie

Andreas Polzer, Stefan Kowalewski
Embedded Software Laboratory, RWTH Aachen
Aachen, Germany
{polzer | kowalewski}@embedded.rwth-aachen.de

Abstract: In this paper, we report on techniques for variability and evolution in Model-based Engineering of Embedded Systems. The techniques are based on an integration of domain-specific languages for embedded systems with model-driven techniques for Software Product Lines. In particular, we discuss (1) product configuration with interactive tools, (2) product derivation with model transformations, and (3) first steps towards feature-oriented evolution planning.

1 Introduction

Since many embedded systems are managed and developed as product lines of similar systems it seems to be natural that techniques for Model-based Engineering of Embedded Systems (MBEES) should be combined with techniques from Variability Modelling and Software Product Lines (SPL) to get the best of both worlds. However, although there are interesting first results in this direction, many challenges remain:

- *Integration* – Despite the fact that appropriate concepts are available in the separate engineering disciplines (MBEES and SPL) the integration of the corresponding techniques and tools into consistent frameworks and tool-chains remains challenging. In particular, when trying to introduce variability and model-driven product line techniques to embedded systems, we have to respect existing domain-specific languages and the corresponding engineering practices, which are established in industry.
- *Lack of support for variability* – Domain-specific modelling languages (DSML) for embedded systems lack support for variability. Hence, to realise variability we have to represent it either internally in the DSML with helper constructs (e.g., custom blocks that are tagged as "variation points") or integrate the DSML with external

variability modelling techniques (e.g., by mapping blocks to features in a feature model).

- *Consistency and semantics of variable models* – For isolated instances of the DSML we usually have semantics (explicitly or implicitly, e.g., through an implementation) and can check if the given instance is a valid model (according to the language definition). However, when dealing with a whole product line it is much more difficult to determine if *all* potential variants are valid models. The problem gets even more complex when we check the fulfillment of requirements (e.g., through formal verification or tests).
- *Complexity handling* – Many authors in variability modelling and software product lines report complexity problems. These problems do not come from technical limitation alone. Product line engineering (PLE) involves many interactive activities and the sheer complexity of the involved artefacts (e.g., the number of variation points and dependencies between them) quickly reach the cognitive limitation of the human engineers.
- *Insufficient efficiency* – Although there has been some progress in model-driven PLE [DRGN07, VG07, HKW08], in real projects the processes are often performed with insufficient efficiency. This is partly caused by lack of proper automated techniques. Another reason is the complexity of model-driven frameworks, e.g., when setting up automated workflows which process he models.

It should be noted that we do not claim that we solve all these challenges. Instead the described issues set the context and objectives for the following discussion.

In the remainder of the paper, we report on our current research in the area of variability and evolution of embedded systems, including interactive product configuration (Section 3), product derivation including the realisation of variability through model transformation (Section 4) and first steps towards model-based support for feature-oriented evolution planning (Section 5). The paper concludes with a brief overview of related work and final thoughts.

2 Overview of Our Approach

Before we go into more detail, we give an overview of our framework. The approach (cf. Figure 1) can be structured along two dimensions. Vertically, we distinguish *Domain Engineering* and *Application Engineering*. This is consistent with well-known frameworks for Software Product Lines (SPL), e.g., [PBvdL05, CN02]. These two levels can be augmented with a third level of *Language Engineering*. Horizontally, we move (from left to right) from abstract *Requirements* over *Features* towards the concrete *Implementation*.

Please note that we mark processes with numbers (❶ to ❺) and artefacts with uppercase letters (🅐 to 🅒). In addition, we use indexes (e.g., $🅐_d$ and $🅐_a$) to distinguish artefacts on Domain Engineering and Application Engineering level.

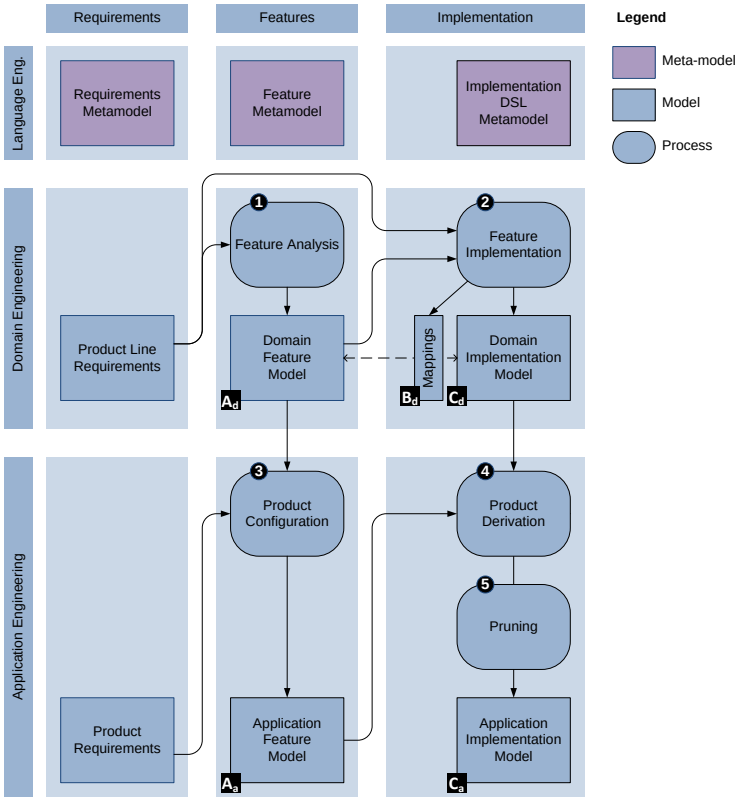


Figure 1: Overview of the workflow with artefacts and processes.

2.1 Domain Engineering

Many SPL approaches use two types of models (see Figure 1): A model describing the available choices, e.g., a feature or variability model A_d and one or more implementation models describing how these choices are implemented C_d. Usually these models are mapped onto each other B_d to support further processing.

The activities during *Domain Engineering* start with *Feature Analysis* ❶ creating a *Domain Feature Model* A_d, which defines the scope and configuration options of the SPL. Subsequently, in *Feature Implementation* ❷ a corresponding implementation is created. This can, for instance, be given in the native Simulink format (`*.mdl` files). To access this in our model-based approach, which relies on Eclipse-based model frameworks, we have to convert it into a model. For this we use techniques based on Xtext [EFb]. (see [PBWK09, BPK09a, BPK09b] for more details) and g. As a result we get the corresponding *Domain Implementation Model* C_d and Feature-Implementation Mappings B_d, which are required as input for *Application Engineering*.

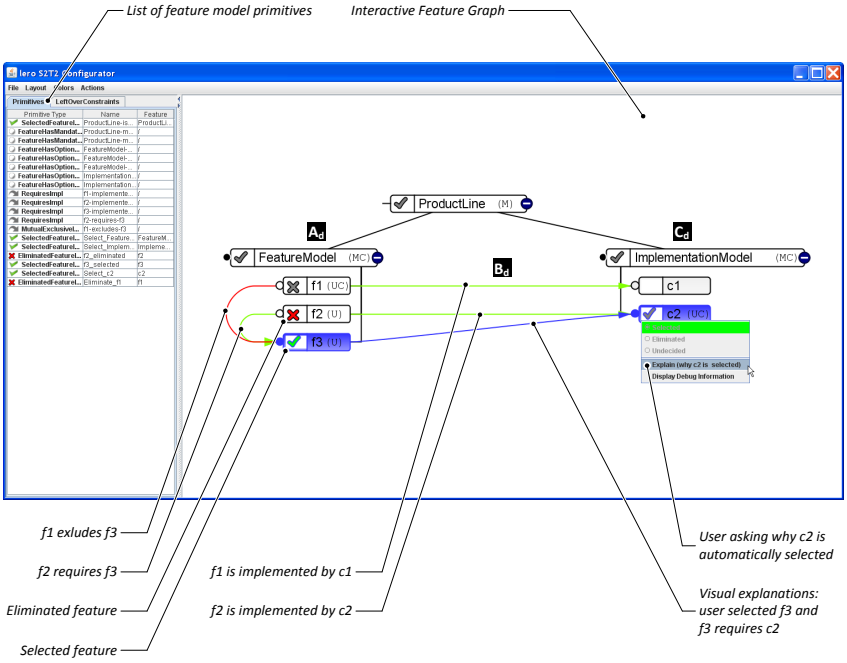


Figure 2: Configuration of a product line model in the S2T2 Configurator.

2.2 Application Engineering

In our framework the activities in *Application Engineering* are grouped in two processes, *Product Configuration* and *Product Derivation*.

In *Product Configuration* ❸ configuration decisions are made, taking into account the product-specific requirements and the *Domain Feature Model* A_d. The result defines the product in terms of features and is saved as the *Application Feature Model* A_a.

In *Product Derivation* ❹ the configuration is turned into a product. The derivation starts with the *Domain Implementation Model* C_d and derives a product-specific implementation. After additional pruning ❺, we get the *Application Implementation Model* C_a, which can be used in further processing steps (e.g., Simulink code generation) to create the executable product. In the following sections, we will look at these processes in more detail.

3 Product Configuration

When configuring a product one major challenge is the complexity of the underlying models, caused, e.g., by the large number of variation points and dependencies between them.

This complexity is less problematic because of technical limitations (e.g., performance of algorithms) but because of the limited cognitive capacity of human engineers during the interactive parts of the process (e.g., when understanding all consequences of a particular configuration choice). One potential solution are approaches that support cognitive tasks that involve complex models, e.g., by applying software visualisation to enable visual configuration and visually-informed variability management [CNP+08]. However, with the objective of configuring a product, just a visual representation is not enough. In addition, we require a formal semantics that defines how configuration options and constraints are to be interpreted and executed during interactive configuration. For instance, to define the consequences of selecting a particular feature.

In our work, we have taken formal semantics of feature models [CW07] and designed an interactive tool, *S2T2 Configurator*, which integrates a formal reasoning engine and a visual interface [BJS09, BSP09]. Figure 2 shows a very simple sample model in *S2T2 Configurator*. The visualisation is optimised to allow interaction with multiple submodels and the mappings between them (corresponding to the models A_d, C_d, B_d in Figure 1).

It should be noted that the shown visual elements for the implementation model C_d are simplified representations, which are shown here for configuration purposes. For instance, this allows the engineer to specify that a certain component is not available at the moment. *S2T2 Configurator* will then update the available features accordingly. These simplified representations can automatically be derived from the real implementation models [BPK09a].

4 Product Derivation

After the product-specific configuration has been determined, the corresponding implementation needs to be derived. In the presented approach, this done in two steps, *Negative Variability* and *Pruning*.

4.1 Negative Variability

The technique of *Negative Variability* ❺ assumes that the Domain Implementation Model C_d contains the *union* of all potential implementations. A product-specific implementation is then derived by selectively copying elements based on the configuration decisions stored in *Application Feature Model* A_s and the Feature-Implementation Mappings B. The corresponding model transformation deletes all elements that are mapped to eliminated features and keeps all elements that are mapped to selected features.

As an example consider the process ❹ in Figure 3, which applies a feature configuration (not shown in the figure) and removes all implementation elements that correspond to eliminated features. In the example, one block for a compass sensor (red block) and one block that implements the compass-based variant of an automatic parking mechanism (one of the blue blocks) are removed.

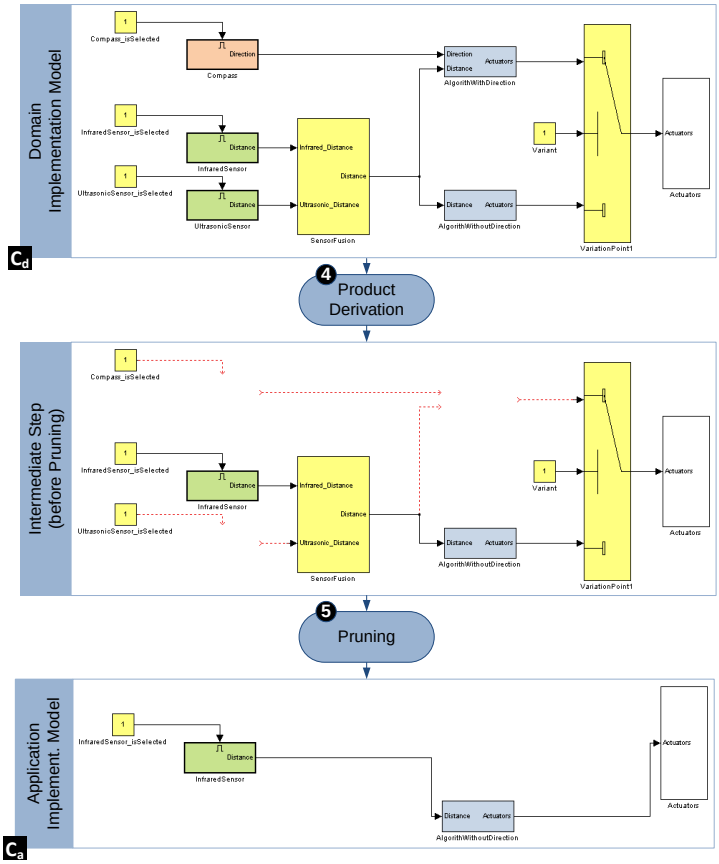


Figure 3: Example of product derivation including pruning.

4.2 Pruning

After the initial negative variability step the resulting implementation model contains dangling elements, e.g., lines that are no longer connected to blocks. Hence, an additional step is required, which we call *Pruning* ❺ (cf. Figure 3). Pruning requires knowledge about the semantics of the particular domain-specific language [BPK09b]. For instance, when deleting a *Block* in Simulink, we have to remove the corresponding *Ports* and as well. These *Ports*, in turn, might be referenced by *Lines* and so on. More details on the approach and the used higher-order model transformations can be found in [BPK09b].

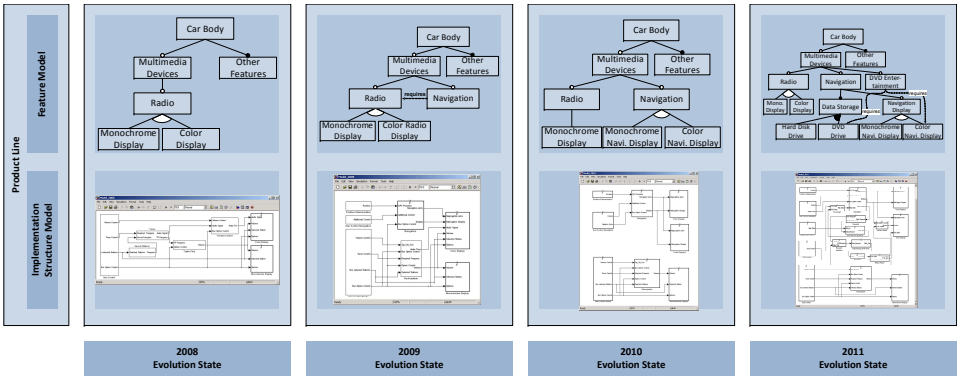
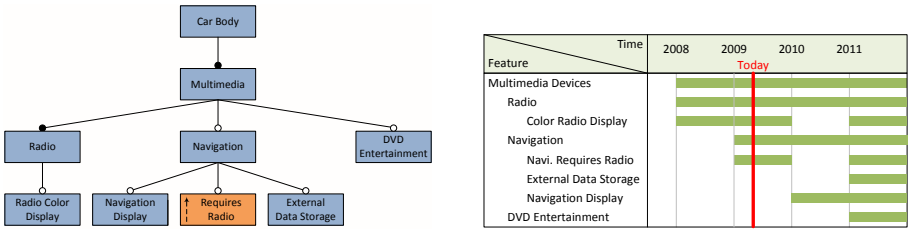


Figure 4: Example product line evolving over time.



(a) Feature model describing choices.

(b) Gantt-chart describing evolution plan.

Figure 5: Example of feature-driven evolution planning with EvoFM.

5 Evolution Planning

Industries that successfully apply product lines often have a strategic perspective and apply long-term planning and evolution of their product portfolios. This is particular true for embedded systems. A typical example is the automotive industry, where an integrated design of hardware and software is applied and requires careful synchronisation of the involved processes.

With *EvoFM* [BPPK09] we strive to provide model-driven support for an feature-oriented planning of evolution. As an example scenario consider the sequence of product lines shown in Figure 4. Each product line (i.e., each evolution step) consists of several inter-related models (e.g., a Domain Feature Model and an Implementation Model as discussed earlier).

Figure 5 gives a first impression of how *EvoFM* can be used to plan evolution. Please note that *EvoFM* does *not* represent a different view on the regular feature models in the product line. Instead it provides an abstract representation of the changes and variability that happen *over time* along the evolution path. For instance, the example EvoFM model shown in Figure 5(a) indicates that the functional cluster *Radio* will remain present throughout the

evolution (hence, it is a mandatory feature in EvoFM), whereas *Radio Color Display*, *Navigation* etc. are subject of evolution and can be introduced or removed (optional feature). Note the special feature *Requires Radio* representing the potential introduction or removal of a dependency. Given this model, evolution steps can be planned as its configurations. A sequence of evolution steps can be represented with a Gantt-Chart visualisation (see Figure 5(b)).

Given a history of past evolution steps, the current product line models, and the evolution plan, we then can derive future instances of product line models. This is not trivial and involves many challenges. For instance, we have to trade-off between an abstract evolution model (easier to create and handle, hard to derive concrete product line models) and an more concrete evolution model with lots of technical details (harder to create and handle, easier to derive concrete product line models).

More details on EvoFM, the underlying framework, and some usage scenarios, can be found in [BPPK09].

6 Related Work

Weiland [WR05] addresses variability in Simulink by marking standard blocks to represent choices. Hence, the implementation model also contains the variability information. A variant is chosen by setting the parameters and selecting a specific signal path. Kubica [Kub07] starts from a feature model in *pure::variants*, where the required features are chosen. Then, the corresponding Simulink model is built automatically from templates and fragments.

There are other techniques for variability in DSML. For instance, Voelter and Groher [VG07] describe how to use openArchitectureWare [ope] for SPL Engineering. They evaluate their approach with a small sample product line of Smart Home applications.

When dealing with variability, a typical challenge is the mapping of features to their implementation. Czarnecki and Antkiewicz [CA05] used a template-based approach where visibility conditions are described in OCL. Heidenreich et al. [HKW08] introduce FeatureMapper, a which can map features to arbitrary EMF-based models [EFa].

7 Conclusions

We would argue that the integration of techniques from (1) model-driven software in the large, software product lines, and variability modelling with (2) model-driven engineering of embedded systems promises great potential. Nevertheless, many challenges and research questions remain, e.g., in the improvement of efficiency, the integration of interactive and automated techniques, and the model-driven support for evolution.

Future work involves the improvement of the model transformations, which implement the product derivation and pruning operations. Moreover, to realise the EvoFM approach we

are currently developing a catalog of evolution operations, which specify how particular evolution changes affect the product line models during an evolution step. Here we see major challenges in the handling of incomplete or inconsistent evolution plans.

Acknowledgements

This work is partially supported by Science Foundation Ireland under grant no. 03/CE2/I303_1 to Lero – The Irish Software Engineering Research Centre, http://www.lero.ie.

References

[BJS09] Goetz Botterweck, Mikolas Janota, and Denny Schneeweiss. A Design of a Configurable Feature Model Configurator. In *Proceedings of the 3rd International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS 09)*, January 2009.

[BPK09a] Goetz Botterweck, Andreas Polzer, and Stefan Kowalewski. Interactive Configuration of Embedded Systems Product Lines. In *International Workshop on Model-driven Approaches in Product Line Engineering (MAPLE 2009), colocated with the 12th International Software Product Line Conference (SPLC 2008)*, 2009.

[BPK09b] Goetz Botterweck, Andreas Polzer, and Stefan Kowalewski. Using Higher-order Transformations to Derive Variability Mechanism for Embedded Systems. In *ACES-MB 2009, 2nd International Workshop on Model Based Architecting and Construction of Embedded Systems, collocated with MODELS*, 2009.

[BPPK09] Goetz Botterweck, Andreas Pleuss, Andreas Polzer, and Stefan Kowalewski. Towards Feature-driven Planning of Product-Line Evolution. In *1st International Workshop on Feature-oriented Software Development (FOSD 2009), collocated with MODELS*, 2009.

[BSP09] Goetz Botterweck, Denny Schneeweiss, and Andreas Pleuss. Interactive Techniques to Support the Configuration of Complex Feature Models. In *1st International Workshop on Model-Driven Product Line Engineering (MDPLE 2009), held in conjunction with ECMDA 2009*, Twente, The Netherlands, June 2009.

[CA05] Krzysztof Czarnecki and Michal Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *GPCE'05*, Tallinn, Estonia, September 29 - October 1 2005.

[CN02] Paul Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns*. The SEI series in software engineering. Addison-Wesley, Boston, MA, USA, 2002.

[CNP+08] Ciaran Cawley, Daren Nestor, Andre Preussner, Goetz Botterweck, and Steffen Thiel. Interactive Visualisation to Support Product Configuration in Software Product Lines. In *Second International Workshop on Variability Modelling of Software-intensive Systems (VAMOS 2008)*, Essen, Germany, January 2008.

[CW07] Krzysztof Czarnecki and Andrzej Wasowski. Feature Diagrams and Logics: There and Back Again. In *SPLC '07: Proceedings of the 11th International Software Product Line Conference (SPLC 2007)*, pages 23–34, Washington, DC, USA, 2007. IEEE Computer Society.

[DRGN07] Deepak Dhungana, Rick Rabiser, Paul Grünbacher, and Thomas Neumayer. Integrated tool support for software product line engineering. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 533–534, New York, NY, USA, 2007. ACM.

[EFa] Eclipse-Foundation. EMF - Eclipse Modelling Framework. `http://www.eclipse.org/modeling/emf/`.

[EFb] Eclipse-Foundation. Xtext. `http://www.eclipse.org/Xtext/`.

[HKW08] Florian Heidenreich, Jan Kopcsek, and Christian Wende. FeatureMapper: Mapping features to models. In *ICSE Companion '08: Companion of the 13th international conference on Software engineering*, pages 943–944, New York, NY, USA, 2008. ACM.

[Kub07] Stefan Kubica. *Variantenmanagement modellbasierter Funktionssoftware mit Software-Produktlinien*. PhD thesis, Univ. Erlangen-Nürnberg, 2007. Arbeitsberichte des Instituts für Informatik, Friedrich-Alexander-Universität Erlangen Nürnberg.

[ope] openarchitectureware.org. Official Open Architecture Ware Homepage. `http://www.openarchitectureware.org/`.

[PBvdL05] Klaus Pohl, Guenter Boeckle, and Frank van der Linden. *Software Product Line Engineering : Foundations, Principles, and Techniques*. Springer, New York, NY, 2005.

[PBWK09] Andreas Polzer, Goetz Botterweck, Iris Wangerin, and Stefan Kowalewski. Variabilitaet im modellbasierten Engineering von eingebetteten Systemen. In *7. Workshop Automotive Software Engineering, collocated with Informatik 2009*, Luebeck, Germany, September 2009.

[VG07] Markus Voelter and Iris Groher. Product Line Implementation using Aspect-Oriented and Model-Driven Software Development. In *11th International Software Product Line Conference (SPLC 2007)*, Kyoto, Japan, September 2007.

[WR05] J. Weiland and E. Richter. Konfigurationsmanagement variantenreicher Simulink-Modelle. In *Informatik 2005 - Informatik LIVE!, Band 2*. Koellen Druck+Verlag GmbH, Bonn, September 2005.