Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 182 (2007) 171–186
www.elsevier.com/locate/entcs

# Combining Product Lines and Model-Based Development

Bernhard Schätz

*Technische Universität München, Fakultät für Informatik, Boltzmannstr. 3, D-85748 Garching bei München, Germany*
*schaetz@in.tum.de*

**Abstract**

Using model-based development has shown to increase efficiency and effectiveness of software production. However, with software as an integral part of products with customized functionalities, explicit treatment of product lines is increasingly becoming necessary to cope with this additional complexity. To combine these aspects, which are generally considered only in isolation, a conceptual model addressing both the aspects of product-line engineering as well as aspects of component systems is introduced, and the consequences concerning product line identification and instantiation are illustrated.

*Keywords:* Product line, variability, model-based, conceptual model, meta model, consistency

## 1 Introduction

As software is increasingly becoming an integral part of various end-customer products, there is a rising demand for reuse and variants of software components to support customized functionalities. To cope with productivity as well as quality issues, systematic approaches to variations of systems are needed, addressing the issues of identifying product lines as well as creating variants.

Classical approaches to product lines – e.g., [7] or [10] – generally only consider features in general abstracting from a detailed model of the system under development; as a result, the definition of a variant of a product line does not automatically result in its effective construction. Furthermore, due to the abstraction, additional abstract dependencies between these features are needed to reflect variation constraints; by decoupling these high-level dependencies from the effective constraints inherit in the detailed model, the identification of possible product lines becomes additionally challenging.

To cope with these issues, an integrated model addressing both the aspects of product-line engineering as well as domain-specific aspects of the description of component systems is introduced and the resulting possibilities concerning product


doi:10.1016/j.entcs.2006.09.038

line identification and instantiation are discussed. As product lines are primarily concerned with methodical aspects of software engineering, the model is introduced from two different points of view: a formal model, allowing to precisely define the aspects of variability in the component-based description of software systems; and a technical model, allowing to illustrate the possible application of these formal concepts.

### *1.1 Motivation*

As stated in [7], product line software engineering targets "the strategic reuse of product line assets (e.g., architectures and software components)" by analyzing "commonalities and variabilities among products". However, approaches like [7] generally analyze and structure features of possible solutions, using uninterpreted features as the corresponding base concept without further relation to the assets. As a consequence, these approaches focus on the analysis of dependencies, however abstracting away from the causes for these dependencies.

On the other hand, model-based development targets the support of the development process using models capturing the concepts of the application domain. While approaches like [8] provide detailed models of the concepts of an application domain as well as their dependencies to construct the description of a system under development, they do not explicitly address the relations between different possible solutions. As a consequence, these approaches focus on the construction of a specific solution without supporting the description of variability.

To combine the benefits of both methods, in the following sections an integrated approach for variability modeling and model-based development is introduced and a possible tool-support is illustrated.

### *1.2 Overview*

In the approach presented here, the combination of product lines and model-based development is achieved by providing *product lines of system descriptions* through *explicitly combining the concepts* used for system descriptions on the one hand and variability descriptions on the other hand.

In Section 2 this combination is achieved by providing a generic model for system descriptions and introducing a model for the description of variability on top of it. Both models are defined both from a formal point of view as well as from a tool-oriented point of view, to illustrate both the underlying intuition as well as a possible tool-support.

In Section 3, the application of this integration is illustrated. To that end, the definition, configuration, and identification of product lines is described on top of the introduced model, both from the formal and user point of view.

Section 4 finally compares the presented approach to related work and discusses further steps.

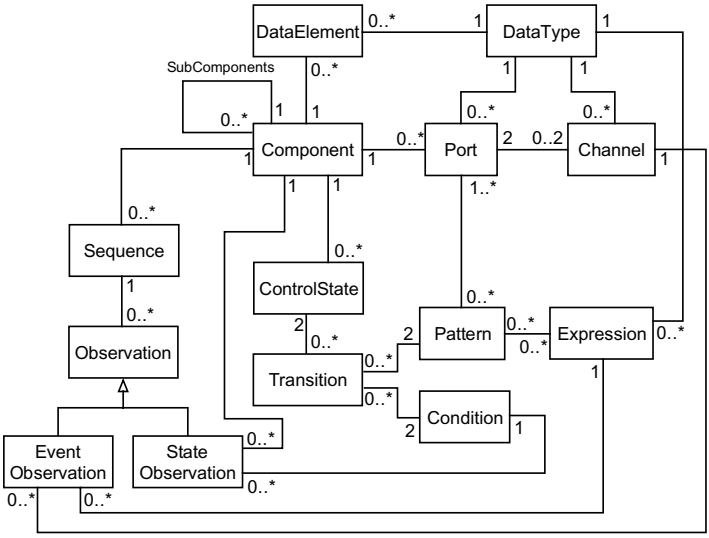


Fig. 1. AutoFocus Conceptual Domain Model (Simplified)

## 2 Modeling Variability of Descriptions

Product lines are applied to construct families of systems within a domain of application. As an integrated approach must describe the aspects of variability *and* the domain of application, a domain model is needed that defines how the description of a system is constructed; furthermore, a variability model is needed that defines the differences and the commonalities between the system descriptions within a product line.

### *2.1 Domain Models*

To construct formalized specifications of a system under development, a 'syntactic vocabulary' or *conceptual model* [9] is needed. This conceptual model [1] consists of the *modeling concepts and their relations* used to construct a description of a system. These modeling concepts are reflected in the techniques used to describe a system; Figures 2 (left-hand side) and 3 show examples of these descriptions.

As shown in Figure 1, in the AutoFocus approach, the conceptual model for the application domain of reactive components includes the elements like

**Components** are units encapsulating data, structure, and behavior, communicating with their environment.

**Data types** define data structures used by components.

**Data elements** like variables provide a means to store persistent state information inside a component.

**Ports** are a component's means of communicating with its environment.

[1] In the context of technologies like the Meta Object Facility, the class diagram-like definition of a conceptual model is generally called *meta model*.

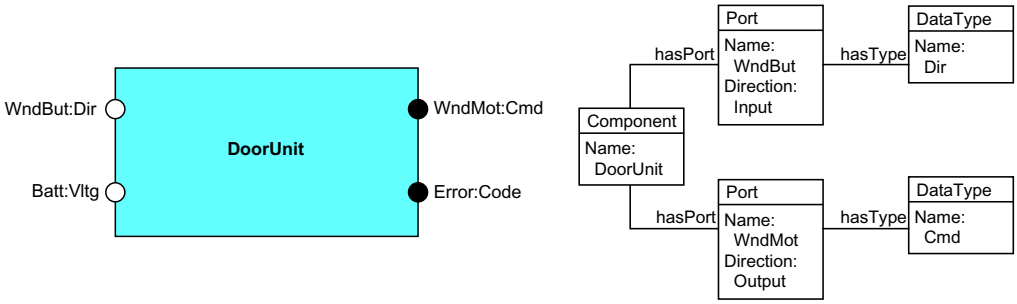


Fig. 2. AutoFocus Component Description and Instance Model (Simplified)

**Channels** connect component ports, defining the communication structure of a system.

**Control States** characterize specific modes of a component, influencing and influenced by its reactions.

**Transitions** between control states define the flow of control within a component, with **Conditions** describing the data state before and after the execution of a transition, and **Patterns**characterizing the messages–input and output–consumed and produced during the execution of a transition.

From a CASE-oriented point of view, the conceptual model is the 'data model' of the products explicitly handled by the tool. The conceptual model can be described as a class diagram (see Figure 1 for a simplified version used in the AutoFocus approach).

To define the notion of a *conceptual model* in the context of product lines more formally, we use the interpretation along the lines of [9]. Intuitively, the conceptual model consists of two classes of elements:

**Entities:** domain-specific concepts used to model a system; typical examples in the AutoFocus conceptual model are the concepts *Component*, *Port*, *Channel*, or *Data Type*

**Relations:** domain-specific dependencies between the modeling entities; typical examples in the AutoFocus conceptual model are the relations *SubComponent*, *hasType*, *hasStartPort*

Entities and relation form the *conceptual domain*. The conceptual domain $\mathcal{C} = (\mathcal{E}, \mathcal{R})$ consists of a collection $\mathcal{E} = (\mathcal{E}_1, \ldots, \mathcal{E}_m)$ of–generally infinite–sets of modeling elements, and a collection of $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_n)$ of relations $\mathcal{R}_i = \mathcal{E}_j \times \mathcal{E}_k$ between these sets of modeling elements. [2]

In case of the AutoFocus conceptual model, examples for sets $\mathcal{E}_i$ are *Component*, *Port*, or *Channel*; typical examples for $\mathcal{R}_i$ are *SubComponents* or *hasStartPort* with $SubComponents = Component \times Component$ and $hasStartPort = Port \times Channel$. Intuitively, the conceptual domain describes the "modeling universe", from which

[2] For reasons of brevity, only binary relations are used in the following.

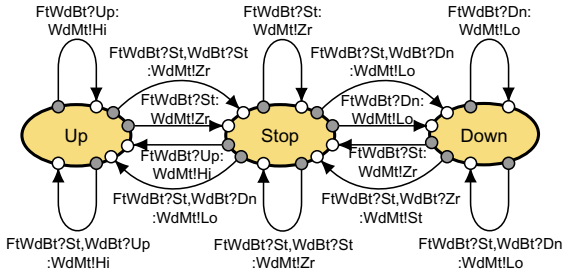


Fig. 3. Behavioral Description of BackWindowControl Component

specific instances of the description of a actual system are constructed. [3]

Based on its conceptual domain, the conceptual model defines the possible system descriptions that can be constructed within this domain. Intuitively, a system description is a "sub-model" of the conceptual domain, with sub-sets of its entities and relations. More formally, a description $(\mathcal{E}', \mathcal{R}') = ((\mathcal{E}'_1, \ldots, \mathcal{E}'_m), (\mathcal{R}'_1, \ldots, \mathcal{R}'_n))$ is a sub-model of $(\mathcal{E}, \mathcal{R}) = ((\mathcal{E}_1, \ldots, \mathcal{E}_m), (\mathcal{R}_1, \ldots, \mathcal{R}_n))$–in short $(\mathcal{E}', \mathcal{R}') \sqsubseteq (\mathcal{E}, \mathcal{R})$–if $\mathcal{E}'_i \subseteq \mathcal{E}_i$ and $\mathcal{R}'_j \subseteq \mathcal{R}_j$.

Figure 2 shows the corresponding entities and relations to describe a control component for a power window embedded into a car door, monitoring a button position and the current battery voltage, controlling the window motor, a possibly producing an error code. The sub-model of the AutoFocus conceptual domain describing this control unit contains, e.g., entities $Component = \{DoorUnit\}$ and $Port = \{WndBut, \ldots, Error\}$, as well as relations $hasPort = \{(WndBut, DoorUnit), (WndMot, DoorUnit)\}$ and $hasType = \{(WndBut, Dir), (WndMot, Cmd)\}$.

As shown in the left-hand side of Figure 2, graphical notations in from of block diagrams are used to describe instance models. From a tool-oriented point of view, these sub-models are described by object diagrams.

To ensure the well-formedness of the system descriptions defined by the conceptual model, *conceptual consistency conditions* are assigned to it [9]. Consistency conditions are defined as restricting properties over the entities and relations of the conceptual domain. Figure 1 includes examples of those restrictions in form of 'arity' constraints imposed on the relations. Examples for those conditions in the AutoFocus domain of reactive components are

- each port must have a defined type, i.e. $\forall p \in Port.\exists t \in Type.hasType(p, t)$
- each channel must have a defined start and an end port, i.e., $\forall c \in Channel.\exists s \in Port.\exists e \in Port.hasStartPort(c, s) \wedge hasEndPort(c, e)$
- each port is end port of only at most one channel, i.e., $\forall p \in Port.\forall c \in Channel.\forall d \in Channel.hasEndPort(c, p) \wedge hasEndPort(d, p) \Rightarrow c = d$

These conditions are either *enforced through construction* or *imposed a specific steps.* The former–generally enforced by supplying only construction operations ensuring

[3] In general the modeling elements of the conceptual domain contain attributes (e.g., *Name*); for sake of brevity, these attributes are ignored in the remainder.

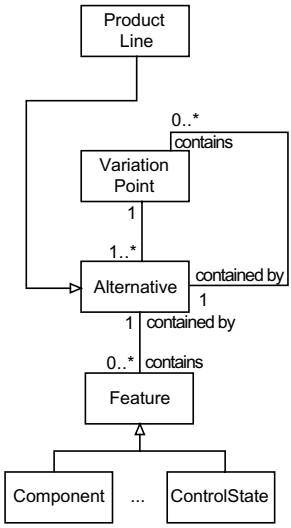


Fig. 4. Conceptual Variability Model

the validity of these conditions–are distinguished between definedness and uniqueness conditions, as illustrated by the examples (defined type of a port, defined start and end port of a channel, unique channel of an end port).

In total, a conceptual model $\mathcal{CM} = (\mathcal{C}, \mathcal{M})$ consists of the conceptual domain $\mathcal{C} = (\mathcal{E}, \mathcal{R})$ describing the entities and relations to construct descriptions, and collection of sub-models $\mathcal{M} \subseteq \{(\mathcal{E}', \mathcal{R}') \mid (\mathcal{E}', \mathcal{R}') \sqsubseteq (\mathcal{E}, \mathcal{R})\}$, each fulfilling the consistency conditions characterizing $\mathcal{CM}$.

### *2.2 Variability Models*

The conceptual model is used to formalize the description of a system under development. However, to describe and exploit the commonalities and differences between variants of system descriptions, additional domain-independent concepts are needed:

**Product Line:** Collection of all variants of a product

**Alternative:** Variable aspects of the variants of a product line

**Variation Point:** Group of related alternatives of a product line

**Variant:** Specific instance of a product line

**Feature:** Distinguishable element of a variant

Concerning the power window functionality, typical instances of the introduced variability concepts are:

**Product Line:** A front door control unit for 2-Door/4-Door, Driver/Passenger, with/without child protection, and with/without pen protection

**Alternatives:** 4-Door-Control (including sensor/actor signals and control functionality for back door, and the possibility to choose child protection), 2-Door-

Control (sensor/actor signals and functionality for front door)

**Variation Points:** Number of doors (2-Door/4-Door), side of door (Driver/Passenger), Child Protection (Included/Excluded)

**Variant:** Number of doors = 4 Door, side of door = Driver, Child Protection = Included

**Features:** Window-Control-Port, Protection-Off-Signal, Movement-Stopped-State

The above interpretation of the variability concepts uses a result-oriented point of view characterizing a product line as a family of products. However, from a construction-oriented point of view, the interpretation of these concepts characterizes the description of product lines:

**Product Line:** Set of all variation points and their associated alternatives

**Alternative:** Set of associated features and (sub-)variation points

**Variation Point:** Set of possible alternatives

**Variant:** Consistent selection of an alternative per variation point

**Feature:** Element of the conceptual domain

Figure 4 shows a conceptual variability model for the description of a product line. Similar to the conceptual domain model, for that purpose an class diagram formalization is used. An alternative aggregates the features of the conceptual domain model.[4] By means of an explicit variation points, a set of alternatives can be combined into sub-alternatives. Thus, a the description of a complete *Product Line* can be understood as a tree with *Alternatives* and *Variation Points* as its nodes; each node has an associated set of *Features* aggregated by it.[5] *Variants* are not directly represented in the description of a product line; instead–as discussed in Subsection 3.2–they define the configuration of descriptions by fixing possible alternatives.

Based on the tree-like description of a product line, reflecting the construction-oriented view, the variability-oriented notions are formalized, resulting in a result-oriented view. Using collections of sets of features as the foundation of this formalization, the base concepts are defined as:

**Product Line:** As a product line $\mathcal{F}$ represents a–generally finite–collection of product descriptions (i.e., models), it can be formalized as $\mathcal{F} = \{F_1, \ldots, F_n\}$ of descriptions $F_i \sqsubseteq \mathcal{C}$.

**Variant:** As a product line represents the collations of all variants, any model $F_i$ of the product line–described by a set of features–is a variant.

Upon these definitions the notions of an alternative and a variation point are formalized, thus obtaining a *mapping from a product line description to a collection of variants* by interpreting each concept as a product line in itself. Obviously the constructions of alternatives and variation points are commutative concerning their

[4] The features like *Component* or *ControlState* are taken from the AutoFocus conceptual model.

[5] Variation Points only aggregate an empty set of features.

sub-variation points and sub-alternatives, resp. Therefore, the binary operators defining alternatives and variation points can be trivially extended to n-ary versions:

**Feature $f$:** A feature can be formalized as a trivial product line $\{F\}$ with $F \sqsubseteq \mathcal{C}$ containing only the unique feature $f$ and its associated relations.

**Alternative $\mathcal{F}_1 \odot \mathcal{F}_2$:** For an alternative comprising collections $\mathcal{F}_1$ and $\mathcal{F}_2$ of feature sets for its aggregated features or associated sub-variation points, its collection $\mathcal{F}_1 \odot \mathcal{F}_2$ of feature sets is defined by $\mathcal{F}_1 \odot \mathcal{F}_2 = \{F_1 \cup F_2 \mid F_1 \in \mathcal{F}_1 \wedge F_2 \in \mathcal{F}_2\}$.

**Variation Point $\mathcal{F}_1 \oplus \mathcal{F}_2$:** For a variation point comprising collections $\mathcal{F}_1$ and $\mathcal{F}_2$ of feature sets for its sub-alternatives, its collection $\mathcal{F}_1 \oplus \mathcal{F}_2$ of feature sets is defined by $\mathcal{F}_1 \oplus \mathcal{F}_2 = \mathcal{F}_1 \cup \mathcal{F}_2$.

The formalization assigns a set of models to each node of the tree describing a product line, using an inductive, bottom-up fashion. The leaf-nodes correspond to–unique–features; the root-node description is assigned the set of all models characterized by the product line description.

The structural arrangement of the nodes coincides with a partial order $\preceq$ of their corresponding sets of models. The order relation $\mathcal{F}_1 \preceq \mathcal{F}_2$, characterizing $\mathcal{F}_2$ as an extended set of models compared to $\mathcal{F}_1$, is defined as

$$\mathcal{F}_1 \preceq \mathcal{F}_2 \stackrel{\text{def}}{=} \exists \mathcal{F}.\mathcal{F}_1 \odot \mathcal{F} \subseteq \mathcal{F}_2$$

with the root corresponding to the set of all variants as its maximal element in the set of all models of this description. Intuitively, the ordering relation states that the *extension* $\mathcal{F}_2$ of a product line $\mathcal{F}_1$ is obtained by adding additional features to all variants of $\mathcal{F}_1$ as well as new variants. Obviously, the least element in this order is the empty product line $\{\emptyset\}$, and the greatest element is the fully variant product line $\{F \mid F \sqsubseteq \mathcal{C}\}$.

This formal definition focuses on describing the *variabilities within a product line* in form of the variants constructed over a given conceptual model through alternatives and variation points. However, for the practical application in form of tool support, a more construction-oriented view focusing on the *commonalities within a product line* is needed. To that end, the *scope* $\mathcal{S} \sqsubseteq \mathcal{M}$ of a product line description is introduced, capturing all features common to a collection of models of its associated product line $\mathcal{F}$ via $\mathcal{S} = \bigcap_{F \in \mathcal{F}} F$. For each node of the tree describing a product line, its scope characterizes the set of its features common to all the associated models. In contrast to the set of models $\mathcal{F}$ assigned to the nodes of a product line description in a bottom-up fashion, the scope $\mathcal{S}$ associated with each node can be constructed in a top-down fashion. The root nodecontains the features common to all variants of the product line, i.e., all features directly aggregated by the top alternative. Any other alternative additionally comprises all features associated to those alternatives higher in the tree-structure of the description of the product line. As variation points do not aggregate additional features, their scope is identical to the scope of their corresponding super-alternative.

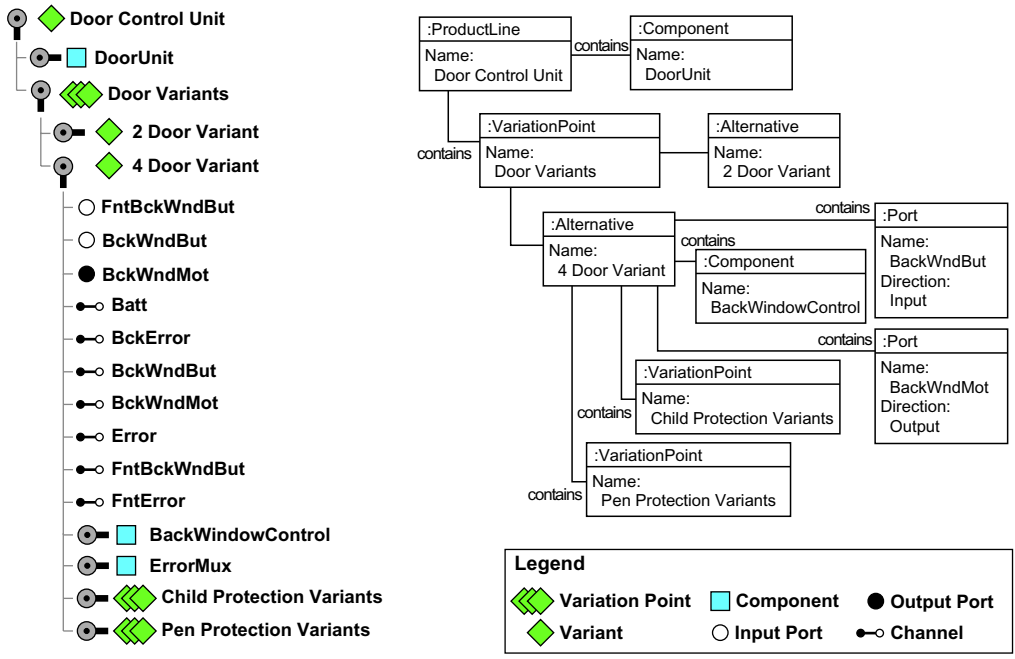


Fig. 5. Product Line Description (with Legend) and Instance Model (Simplified)

As the scopes of a product line form a semi-lattice structure, with alternatives as its elements, a tree-like representation is used to describe a product line and its alternatives (e.g., Door Control Unit, 2 Door Variant), as shown in the left-hand side of Figure 5. A variation point, relating a set of alternatives, is explicitly represented in the tree (e.g., Door Variants, Child Protection Variant). A tree browser–used in the AutoFocus approach to present tree-like models–shows a linearized version of the tree (partly with folded branches like Child Protection Variants).

Besides the variability aspects, the description of a product line also contains the associated features. As shown in the left-hand side of Figure 5, for each alternative, the features introduced by it are listed; e.g., port FntBckWndBut, channel BckError, or sub-component ErrorMux. As defined by the conceptual model, a product line includes domain elements. E.g., BckWndBut is part of the 4 Door Variant. As system descriptions, constructed from the domain model, can be organized in a tree-like structure, elements like the component DoorUnit form the root of a sub-tree of associated features, as shown in the left-hand side of Figure 5.

Note that–as indicated by the description of a product line in Figure 5– for an optimized representation it is sufficient to only explicitly capture those features of a scope that are added by a child-alternative compared to its parent alternative. This optimized representation can be used to supply adequate CASE-support for an integrated model for variability and domain-oriented aspects.

The right-hand side of Figure 5 shows the structure of the product line in terms of these classes for the DoorUnit, with *VariationPoint* Door Variant and associated *Alternatives* 2 Door Variant and 4 Door Variant; the latter again has *VarationPoints* Child Protection Variants and Pen Protection Variants.


Fig. 6. Interface of 4 Door Variant

To support tool-application, a conceptual variability model for a specific domain is constructed. To that end, the conceptual domain model–shown in 1–is integrated into the conceptual variability model–shown in Figure 4. Obviously, for that purpose, selected conceptual consistency conditions imposed on the domain-specific model are weakened to enable the definition of alternatives. For example, a specific end port may be linked to several channels from different alternatives.

However, the description of product lines should result in the characterization of *consistent variants*, i.e., contain only descriptions $P_i \in \mathcal{M}$. Therefore, as discussed in Section 3, the consistency conditions imposed on the conceptual domain model are imposed during the description of a product line or the configuration of a variant, thus ensuring the description of consistent variants.

# 3 Applying Variability

As mentioned in Section 1, the central purpose of production lines is to support

- the systematic definition of differences and commonalities between system descriptions
- the creation of specific description variants of a product line
- the identification of product lines by means of establishing variation points

In this section, corresponding functionalities to meet these purposes are introduced to demonstrate the application of this approach. Besides describing these functionalities from a formal point of view in terms of the model introduced in the previous section, the realization of these formalizations within the AutoFocus tool framework is sketched to illustrate their applicability.

## *3.1 Describing Product Lines*

Obviously, the basic application of an integrated model for variability and domain-specific aspects is the construction of a product line of system descriptions, identifying the commonalities and differences between these descriptions in a structured manner. As shown in the previous section, in principle a product line of system descriptions can be defined using a tree-like representation of variants, variation points, and their associated features. However, as shown in Figure 2, in state-of-the-art CASE-supported approaches, system descriptions are given in from of a collection of graphical representations like component diagrams or state-transition

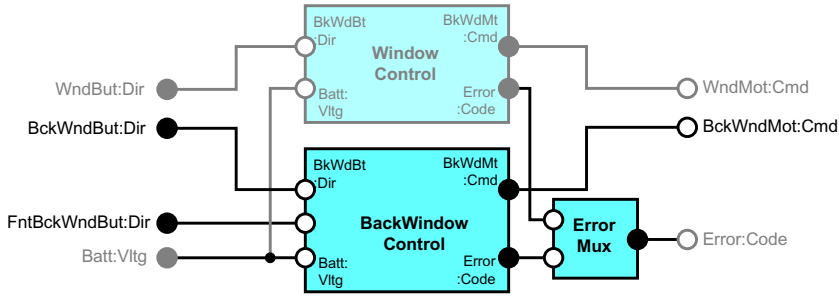


Fig. 7. Internal Structure of 4 Door Variant

diagrams.

Therefore, to support the description of commonalities and differences between alternatives on the diagrammatic level, a separation of concern based on the scopes of alternatives is introduced. As the scope of an alternative specifies the features common to all models of it and thus available within the context of this alternative, the diagrammatic descriptions can be restricted to those features. As furthermore the scopes of a product line description form a semi-lattice in congruence with the structure of the description, features are classified as those aggregated by the current alternative (and thus accessible), by its super-alternatives (and thus visible), and by its sub-alternatives (and thus invisible).

Figures 6 and 7 illustrate this technique, using the example of Figure 5, to construct the component diagram description of the 4 Door Variant alternative of the Door Variants variation point associated to product line Door Control Unit. When constructing the corresponding diagram of this variant as shown in Figure 6, the new (accessible) features like port BckWndBut or BckWndMot describing the extended interface of the control component are added to the original description given in the left-hand side of Figure 2. The (visible) features of the parent description, like ports WndBut or Error are grayed out, to show their independence of the current extension.

Of course, as shown in Figure 7, this principle also applies to diagrammatic descriptions with overlapping views, as used for the description of the external interface and the internal structure of a component.

While scopes were basically introduced as arbitrary collections of features, for construction the description of a product line *scopes are restricted to models* out of M of CM. By interpreting these conditions over the universe defined by the scope of an alternative, the conceptual consistency conditions are enforced on the level of the alternatives. Imposing definedness conditions on scopes of all alternatives of a product line description defines a *sufficient* criterion for the validity of these conditions for the variants of a product line.

In contrast, uniqueness conditions imposed on the scopes of the alternatives of a product line description only define a *necessary* criterion for the validity of these conditions for the variants. As an alternative may comprise several variation

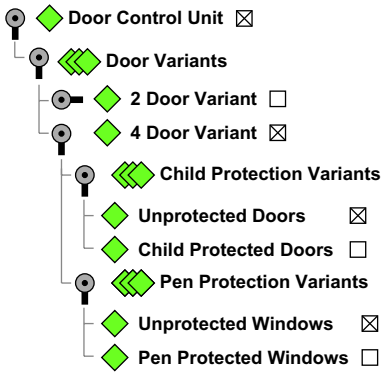


Fig. 8. Configuration of Variant with 4 Unprotected Doors

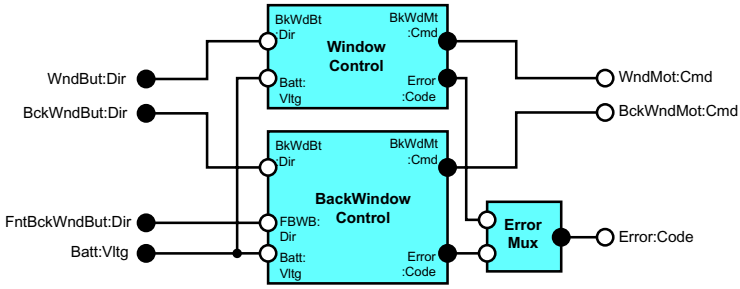


Fig. 9. Configured Variant with 4 Unprotected Doors

points, local uniqueness conditions may get violated globally, e.g., by independently introducing channels for a common end port. Therefore, to ensure consistency for both classes of conditions for all variants, the configuration of product lines is restricted, as described in Subsection 3.2. By this means, the construction of a product line with consistency models is ensured.

### *3.2 Configuring Product Lines*

To effectively apply product lines, variants – reflecting a specific instance of a product line – are formed to obtain a system description containing only features without any variability aspects. As a product line description is formalized as a collection of models, from the formal point of view, this corresponds to the selection of an specific model of this collection.

As in the tool-oriented view with its optimized representation features are only aggregated by the alternatives explicitly introducing those features, here a variant is identified by combining all the alternatives on the path in the semi-lattice of the product line from the root to the maximal element corresponding to the variant.

An essential aspect of product lines concerning the efficient reuse of commonalities and differences in system descriptions is the identification of independent variation points. Therefore, as shown in Figure 8, in the optimized representation offered by the technical view, several independent variation points like Child

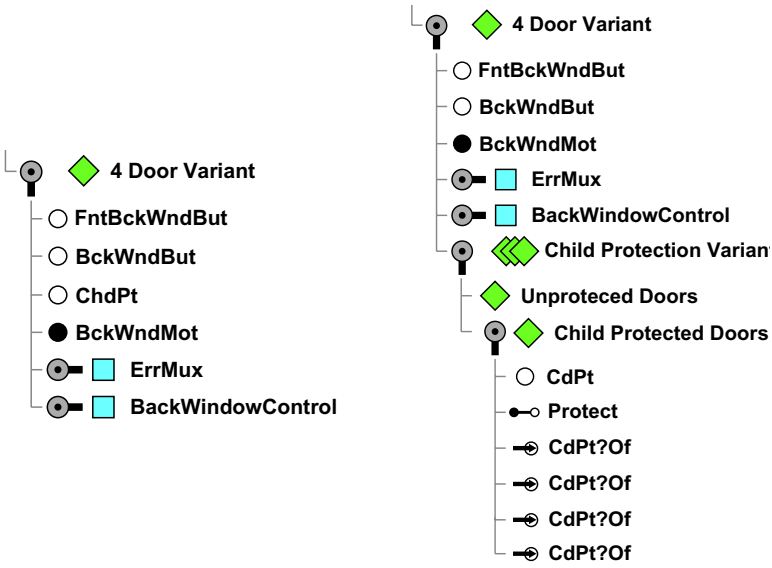


Fig. 10. Identifying the Child Protection Variation Point

Protection Variants and Pen Protection Variations may be aggregated by the same alternative like 4 Door Variant.

Therefore, in that general case, in the tree-like representation of the technical view of a product line, concurrent paths must be selected. Figure 8 illustrates this selection process, showing the selection of alternatives Unprotected Doors and Unprotected Windows, resp., for the independent variation points Child Protection Variants and Pen Protection Variations.

To ensure the configuration of valid variants, the selection of alternatives is restricted to those respecting the imposed consistency conditions. As shown in Figure 9, on the diagram level of the system description, the selection of a specific configuration corresponds to the collection of diagrams without any variability aspects, respecting the conceptual consistency conditions imposed by the domain model. AutoFocus offers a mechanized check to insure the consistency of possible variants.

### *3.3 Identifying Product Lines*

In general, product lines are not developed from scratch but evolve in a stepwise fashion. Obviously, the central step during the identification of a product line is the identification of the differences and commonalities between its variants. Methodically, this corresponds to the – repeated – introduction of a new variation point into the product line including its associated alternatives and the features aggregated by them. Formally, this is equivalent to introducing a new intermediate alternative between two existing alternatives according to the sub-model ordering.

Figure 10 illustrates the introduction of a new intermediate alternative by adding the Child Protection Variants variation point to the Door Control Unit product line from the tool-oriented point of view. Here, e.g., the originally constructed description of a 4 Door Variant alternative – shown in the left-hand side – included a child

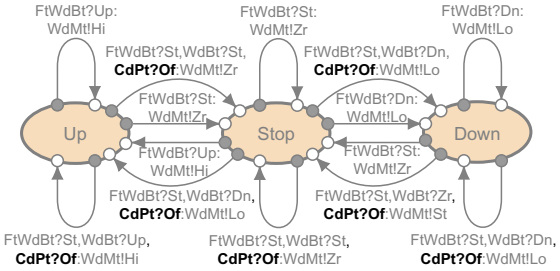


Fig. 11. Identifying Variation Behavior

protection functionality, requiring a corresponding CdPt port to activate and deactivate it. To make this functionality optional, this description is assigned to a new alternative Child Protected Doors, allowing to introduce other alternatives of a new Child Protection Variants variation point; the right-hand side of 10 shows this introduction including a trivial Unprotected Doors alternative.

Note that–as defined by the consistency conditions–the conceptual model imposes certain closure properties when constructing a new intermediate alternative in the semi-lattice of the product line. As shown in the right-hand side of Figure 10 for the tree-like representation and in Figure 11 for the diagrammatic representation, according to the imposed consistency conditions, removing the port CdPt from the 4 Door Variant alternative and assigning it to the newly introduced Child Protected Doors alternative implies to adapt, e.g., behavioral descriptions relying on this port. Thus, as imposed by the consistency conditions and detectable by automatic analysis, the input patterns CdPt?Of used in the state-transition diagram describing the behavior of component BackWindowControl must also be re-assigned to the Child Protected Doors alternative prior to re-assigning port CdPt to the Child Protected Doors alternative.

## 4 Conclusion and Related Work

The approach presented here introduces an integrated model for both variability and domain-specific aspects by explicitly introducing variability into system descriptions.

Approaches like [7] or [10] only focus on the modeling of the variability aspects, without integrating domain-specific aspects, and thus cannot really provide a precise definition or methodical treatment of product lines of system description. On the other hand, approaches like [8] only focus on the modeling of the domain-specific aspects, leaving out all variability aspects. In contrast, here both aspects are integrated into a precisely defined common model, thus supporting the direct description of dependencies between variability constraints on the domain level.

More integrated approaches like [3] or [2] offer extended notations by adding variability aspects into modeling notations, however, without covering concept-rich domain models or addressing the methodical aspects of defining, instantiating, and

identifying product lines. In contrast, here a canonical extension of conceptual domain models is introduced, including the resulting possibilities for using variability.

Closest to the presented approach, [1] and [6] explicitly link variability and domain concepts. In [1], however, unlike in this approach, these links are introduced on a rather informal level; in [4], these links are introduced in form of presence conditions. Thus both approaches do not use an explicit integrated conceptual model. As a result, they do not offer a view-based construction process on the diagrammatic level, supporting immediate application of domain-specific consistency conditions to models with variability.

A similar formalization of the notions of product line, alternative, and variation point was independently and simultaneously developed in [5], based on the general framework of idempotent semi-rings. However, while there the focus is put on the formalization and its properties, here the application of such a formalization on tool-supported construction of product lines is discussed.

To demonstrate its feasibility, the presented approach is currently implemented within the AutoFocus tool framework. To cover all aspects of domain modeling, aspects like the integration of user requirements must be added to the approach. Finally, to assess its adequacy, aspects like the restrictiveness of the conceptual model concerning the possible dependencies between features must be evaluated in industrial case studies.

## Acknowledgement

Its a pleasure to thank Peter Braun, Jan Phillips, and Oscar Slotosch for stimulating discussion; special thanks go to Elmar Jürgens for his critical and constructive remarks to earlier versions of this paper.

## References

[1] Atkinson, C., J. Bayer and D. Muthig, *Component-Based Product Line Development: The KobrA Approach*, in: P. Donohoe, editor, *Software Product Line Conference*, 2000, pp. 289–309.

[2] Bühne, S., G. Halmans and K. Pohl, *Modelling Dependencies between Variation Points in Use Case Diagrams*, in: C. Salesini, B. Regnell and E. Kamsties, editors, *9th Workshop in Requirements Engineering - Foundations for Software Quality*, 2003.

[3] Cengarle, M. V., P. Graubmann and S. Wagner, *Semantics of UML 2.0 Interactions with Variabilities*, in: *2nd International Workshop on Formal Aspects of Component Software*, UNU-IIST Report No. 333, 2005.

[4] Czarnecki, K., C. H. P. Kim and K. T. Kalleberg, *Feature Models are Views on Ontologies*, in: *Software Product Line Conference* (2006).

[5] Höfner, P., R. Khédri and B. Möller, *Feature Algebra*, in: J. Misra, T. Nipkow and E. Sekerinski, editors, *FM 2006: Formal Methods, 14th International Symposium on Formal Methods*, Lecture Notes in Computer Science **4085** (2006), pp. 300–315.

[6] Krzysztof Czarnecki and Michael Antkiewicz, *Mapping features to models: A template approach based on superimposed variants*, in: R. Glueck and M. Lowry, editors, *GPCE 2005 - Generative Programming and Component Engineering* (2005), pp. 422 – 437, lNCS 3676.

[7] Kwanwoo Lee and Kyo C. Kang, *Feature Dependency Analysis for Product Line Component Design*, in: J. Bosch and C. Krueger, editor, *Software Reuse: Methods, Techniques and Tools: 8th International Conference*, LNCS 3107 (2004), pp. 69–85.

[8] Schätz, B., *Mastering the Complexity of Embedded Systems - The AutoFocus Approach*, in: F. Kordon and M. Lemoine, editors, *Formal Techniques for Embedded Distributed Systems: From Requirements to Detailed Design*, Kluwer, 2004 .

[9] Schätz, B. and F. Huber, *Integrating formal description techniques*, in: J. M. Wing, J. Woodcock and J. Davies, editors, *FM'99 – Formal Methods, Proceedings of the World Congress on Formal Methods in the Development of Computing Systems, LNCS 1709* (1999), pp. 1206–1225.

[10] Sochos, P., M. Riebisch and I. Philippow, *The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines*, in: *13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*.