Karlsruhe Institute of Technology

Test, Validation, Analysis

Abstract Change Descriptions for Evolving Delta-Oriented Product Lines

Master's Thesis
of

Marvin Schäfer

at the Department of Informatics
Institute of Information Security and Dependability
Test, Validation and Analysis (TVA)

Reviewer:	Prof. Dr.-Ing Ina Schaefer
Second Reviewer:	Prof. Dr. Ralf Reussner
Advisors:	M.Sc. Philip Ochs
	M.Sc. Dirk Neumann

Completion period: 08. November 2025 – 05. June 2026

Zusammenfassung

In delta-orientierten Produktlinien werden gemeinsame Artefakte über Varianten hinweg wiederverwendet. Wenn sich eine delta-orientierte Produktlinie im Laufe der Zeit weiterentwickelt, können folglich Änderungen, die im Rahmen der Entwicklung in einer Variante eingeführt werden, auch andere Varianten betreffen. *Higher-Order Delta-Modelling* erfasst die Evolution einer delta-orientierten Produktlinie als Ganzes, beschreibt jedoch nicht explizit, welche Änderungen auf der Ebene einzelner Varianten auftreten und wie sich diese Änderungen auf den gesamten Variantenraum auswirken. Dadurch wird es erschwert, den Umfang einer Evolution zu bewerten. Darüber hinaus sollte die Änderungsbeschreibung aufgrund der Natur cyber-physischer Systeme über die beteiligten Domänen hinweg verständlich sein, was eine modellunabhängige Änderungsbeschreibung erfordert. Diese Arbeit begegnet dieser Herausforderung durch die Einführung modellunabhängiger Änderungsdeskriptoren für die Evolution delta-orientierter Produktlinien, welche aus zwei Teilen besteht: Erstens führen wir zur Beschreibung der Evolution einzelner Modellvarianten den *variant change descriptor* ein, für den wir eine formale Definition sowie eine entsprechende Implementierung zur automatisierten Generierung bereitstellen. Zweitens schlagen wir zur Beschreibung der Evolution des Modellvariantenraums den *variant space change descriptor* vor, für den wir eine aggregationsbasierte und eine direkte Berechnungsdefinition sowie Implementierung bereitstellen. Unsere Evaluation bewertet die Korrektheit der variantenspezifischen Änderungsberechnungen und die technische Umsetzbarkeit des *variant change descriptor* und des *variant space change descriptor*. Außerdem vergleicht sie die aggregationsbasierte und die direkte Berechnungen unter Berücksichtigung der Frage, ob redundante Delta-Operationen einbezogen wurden und ob die Gewichtung jeder Modellvariante in beiden Berechnungen denselben Einfluss hat.

Abstract

In delta-oriented product lines, shared artefacts are reused across variants. Consequently, when a delta-oriented product line evolves over time, for the purpose of development, changes introduced in one variant may also affect other variants. *Higher-Order Delta-Modelling* captures the evolution of a delta-oriented product line as a whole, but does not explicitly describe which changes occur at the level of individual variants and how these changes affect the overall variant space, which makes it difficult to assess the scope of an evolution. Furthermore, due to the nature of cyber-physical systems, the change description should be understandable across the domains involved, which requires the change description to be model-independent. This thesis addresses this challenge by introducing model-independent change descriptors for delta-oriented product line evolution, which comprises two parts: Firstly, to describe the evolution of individual model variants, we introduce the *variant change descriptor*, for which we provide a formal definition and an according implementation for automated generation. Secondly, to describe the evolution of the model variant space, we propose the *variant space change descriptor*, for which we provide an aggregation-based and a direct calculation definition and implementation. Our evaluation assesses the correctness of the variant-specific change calculations and the technical feasibility of the *variant change descriptor* and the *variant space change descriptor*. It also compares aggregation-based and direct calculations, considering whether redundant delta operations were taken into account and whether the weighting of each model variant has the same impact in both calculations.

Contents

List of Figures

List of Tables

Acronyms

SGE	System Generation Engineering
PL	Product Line
SPL	Software Product Line
UML	Unified Modeling Language
SysML	Systems Modeling Language
ECU	electronic control unit
HMI	Human-Machine Interface
CAN	Controller Area Network
BCS	Body Comfort System
AV	Attribute Variation
PV	Principle Variation

1. Introduction

Modern systems are increasingly being developed as Product Lines (PLs), which are families of related products that share a common core but differ in specific functions [2]. Product lines are divided into problem and solution space. The problem space models the theoretical variability of the PL and considers the perspectives of stakeholders regarding their problems, requirements and views of the entire domain and individual products. It consists of features and their dependencies. A feature describes user-perceivable functions of a system, and a configuration describes a set of selected and deselected features. In the solution space of a PL, model variants are derived from configurations and, as such, represent realization artefacts from an engineering perspective. The implementation artefacts can be modelled using different modelling languages, such as Unified Modeling Language (UML) and Systems Modeling Language (SysML).

Delta-Modelling is an approach for deriving an individual model variant, based on a specific configuration [16]. Rather than representing each model variant explicitly, a delta model captures variability through deltas, which are modular transformations that describe how to derive model variants from a core model variant [16].

Product lines evolve over time, for example due to changing requirements. *Higher-Order Delta-Modelling* extends delta modelling by allowing deltas themselves to be modified by higher-order deltas [10]. This enables the modelling of changes over time, thereby offering a variability-aware concept to represent the evolution of a delta-oriented PL.

Model-specific changes, represented by Delta-Modelling, can be made more human-understandable across PL developers by model-independent *abstract change descriptors* [15]. A change descriptor classifies a given delta operation into a model-independent description language. To describe a delta with an abstract change descriptor, the change descriptors can be aggregated into an *aggregated change descriptor* [15]. Examples of a model-independent description language includes the description model of System Generation Engineering (SGE) by Albers et al. [1], which classifies changes to delta operations as either a carryover variation, Attribute Variation (AV), or Principle Variation (PV) and describes changes to deltas as a share of changed modelling elements.

Change descriptors and aggregated change descriptors can be used to describe changes to models in terms of deltas and delta operations, independently of the model itself. However, this approach purely is not applicable to delta-oriented PLs evolved by higher-order deltas, since changes to individual model variants are only implicitly represented by a higher-order delta. Furthermore, the current scope of abstract change descriptors does not cover the model-independent description of changes to the variant space. This is because they only describe changes to deltas, which provide no information on their impact on the variant space.

In this thesis, we propose a concept to describe the induced change of higher-order deltas on PLs. We explicitly acknowledge changes to individual model variants as well as to the model variant space. Essentially, we want to address the following sub-goals, illustrated in Figure 3.1.

SG1: Describing Change of Individual Model Variants, Introduced by Higher-Order Deltas

We propose a concept to derive the evolution of individual model variants after applying higher-order deltas to a delta-oriented PL. We describe the changes to individual model variants using a *variant change descriptor*, which aggregates all change descriptors for a specific model variant. This preserves the changes semantics and makes them more human-understandable to PL developers. To achieve this, we will use change descriptors within the description model of SGE by Albers et al. [1]. The *variant change descriptor* is thus a mapping of the changed delta operations to the variation types, stating the share of changed modelling elements and the contribution of each variation type. This analysis is useful for product development because it enables the changes in each model variant to be tracked individually [1].

SG2: Describing Change of Model Variant Space, Introduced by Higher-Order Deltas

SG2a We describe the changes made to the model variant space of a PL as a whole. We aggregate the variant change descriptors derived from SG1, referring to the aggregation as a *variant space change descriptor*. This will provide information on the changes made to the entire model variant space, for example the share of occurred variation types within the description model of SGE. In general, aggregating variant change descriptors will only provide an upper bound for the proportional changes in model variants within the model variant space, since model variants can contain redundant delta operations. To aggregate variant change descriptors, we take preconditions or precalculations to the delta model and the higher-order delta into account. This contribution allows PL developers to analyse the extent to which a higher-order delta impacts the PL, giving them the ability to adjust their changes beforehand and reduce the effort required for the PL evolution.

SG2b As the number of model variants increases exponentially with the number of deltas, the realization of SG2a becomes increasingly more difficult. Thus, we aim to find description of changes in the model variant space without examining each model variant individually. We achieve this by only using the higher-order delta and the delta model itself to directly calculate the changes in the model variant space. Building on the existing description approach these changes are described and referred to as a *variant space change descriptor*.

Chapter 2 introduces the fundamentals required to understand the design concepts presented in Chapter 3. Chapter 4 explains how the design concepts are implemented. Chapter 5 evaluates and discusses the concepts in relation to a subject system. Chapter 6 compares the focus of related work with that of our thesis, while Chapter 7 concludes the thesis and provides an outlook on future work that builds on the concepts introduced.

2. Basics

This chapter introduces the basic terms and concepts necessary for understanding the design principles of this thesis. Chapter 2.1 introduces PLs, while Chapter 2.2 introduces delta-modelling. Both of these concepts aim to improve the management of variability in product development. Chapter 2.3 covers higher-order delta-modelling, which enables variability to be modelled over time. Finally, change Chapter 2.4 introduces change descriptors, which make model-dependent changes more understandable.

2.1 Product Lines

PL Engineering is an approach that aims to efficiently create a family of related systems by exploiting their commonalities while systematically managing their variability [2]. The primary motivation behind PLs is to improve productivity, reduce development costs, and enhance product quality by enabling systematic reuse. Instead of developing each product independently, organizations develop a shared platform that contains reusable components, architectures, and artifacts. The following description of PLs is based on the work of Apel et al. [2].

A central concept in PLs is the notion of features. Features represent end-user-visible characteristics or system properties that distinguish one product variant from another. Features can describe functional requirements (e.g., support for a specific file format) as well as non-functional properties (e.g., security levels or performance options). To systematically represent the relationships between features, feature models are commonly used. Feature models describe which features are mandatory, optional, or mutually exclusive and define constraints between them. These models form the basis for configuring valid product variants. Developers can derive a specific product from the PL by selecting a set of compatible features, also called a configuration.

Typically there are two different perspectives in the development of PLs as described in the following two sections.

2.1.1 Problem Space

The problem space models the theoretical variability of the PL and considers the perspective of stakeholders regarding their problems, requirements and views of the entire domain and individual products. It consists of features and their dependencies.

2.1.2 Solution Space & Model Variants

The solution space models the implementation artefacts and their dependencies, by deriving them from configurations. A product variant has different domains that must be modelled. Implementation artefact can be modelled using different modelling languages, such as UML and SysML, to represent realisation artefacts from an engineering perspective. An implementation artefact from a specific domain is a model variant of the product variant. Therefore, a complete product variant can be derived from all its model variants. The concept of this thesis lies within the solution space.

2.2 Delta-Modelling

Delta-modelling is a variability modelling approach used in Software Product Line (SPL) engineering to systematically describe and implement differences between product variants [16]. Rather than encoding all variants within a single artifact, delta-modelling represents a core and a set of transformations, called deltas, which modify the core to derive different products. The core represents a valid product that contains the common functionality shared by all products in the PL. Delta-modelling is therefore a common solution for modelling SPLs in the solution space, since model variants are indirectly modelled within the delta model and can also be derived from it.

A delta δ encapsulates a set of delta operations (e.g. $\delta = \{op_1, op_2, op_3\}$). Delta operations atomically manipulate elements of the model. The operations include:

- Addition operation (add e/r), which adds new elements e or relations r to the model.
- Removal operation (rem e/r), which deletes existing elements e or relations r from the model.
- Modification operations (mod e), which change properties or internal structures of existing elements e from the model.

Each delta specifies how the model changes when a specific feature is selected. A delta sequence σ is a specific combination of deltas (e.g. $\sigma = \{\delta_1, \delta_2, ..., \delta_k\}$). By applying a delta sequence σ to the core, a product variant can be derived. Each product variant can be derived using at least one specific delta sequence.

Each delta is associated with application conditions to specify under which feature selections a delta should be applied. In addition, delta-modelling may define ordering constraints between deltas. Since multiple deltas can modify the same parts of a model, the order in which they are applied can affect the resulting variant. Ordering constraints ensure that transformations are applied in a consistent and valid sequence. For example, a delta that modifies a component may require that another delta adding this component has already been applied.

2.2.1 Monotonicity

Flexibility provided by delta operations can lead to *add* and *rem* operations mutually cancelling each other. Monotonicity is a property for delta models that forbids mutual *add* and *rem* operations. Increasing monotonicity is when a product variant can only be constructed by adding new content to the core. On the other hand, decreasing monotonicity is when a product variant can only be constructed by removing content from the core.

Damiani et al. [6] defines three different degrees of monotonicity:

- Strictly-Increasing Monotonic: A Delta Model is strictly-increasing monotonic if it only contains *add* operations.
- Increasing Monotonic: A Delta Model is increasing monotonic if it only contains *add* operations and *mod* operations that only extends existing elements and does not change the already existing body of the element.
- Pseudo-increasing Monotonic: A Delta Model is strictly-increasing monotonic if it does not contain *rem* operations.

2.3 Higher-Order Delta-Modelling

Higher-Order Delta-Modelling (HOD) extends delta-modelling by introducing transformations that operate on deltas themselves [10]. While traditional delta-modelling defines transformations that modify elements of a core to derive product variants, higher-order deltas modify deltas. This mechanism enables evolution of delta-oriented PLs by allowing changes to variability definitions to be expressed as modular transformations. The following description and definitions of higher order delta-modelling is based on the work of Lity et al. [10].

A higher-order delta $\delta^H = \{op_1^H, ..., op_r^H\} \subset OP^H$ is a transformation that modifies a delta model. Applying a higher-order delta to a delta model results in a modified delta model in which deltas or their contained actions may be added, removed, or modified.

Higher-order deltas contain evolution operations $op^H \in OP^H$, which describe the concrete modifications applied to deltas. These operations correspond conceptually to the operations used in standard delta-modelling but operate on elements of deltas rather than on elements of a model. OP^H represents the set of all evolution operations that can be applied on deltas.

The evolution operations $op^H \in OP^H$ are defined as follows:

- Addition operation (add δ): Introduces a new delta or a new delta operation within an existing delta.
- Removal operation (rem δ): Removes an existing delta or a delta operation from a delta.
- Modification operation (mod(δ, {*add op*,...,*rem op*′,...})): Changes the structure or parameters of an existing delta operation inside a delta.

The application of higher-order deltas occurs before the standard delta application phase. First, higher-order deltas are applied to the delta model, producing an evolved delta model. Afterwards, the resulting delta model is used to derive products by applying the selected deltas to the core product according to the product configuration and the ordering constraints defined in the delta model.

2.4 Abstract Change Descriptor

A common problem when evolving PLs is that model changes are often highly specific to the domain, making them difficult for engineers who do not specialise in this domain to understand [15]. In the following, Och et al. [15] introduces abstract change descriptors that preserves the semantic information of the changes while decoupling it from the model-specific language.

The description model of SGE defines three types of variation: Carryover Variation (CV), Attribute Variation (AV) and Principle Variation (PV). With an atomic mapping function, delta operations are mapped to the AV or PV of the description model of SGE, generating a *change descriptor*. Additionally, a composed mapping function is defined as the mapping of a delta to a metric representing the development risk, generating a *aggregated change descriptor*. Thus, the composed mapping takes each delta operation contained in a delta, representing the change of an underlying model, and computes it into the *aggregated change descriptor*. The development risk is then described with the share of changed modelling elements, which is the ratio of the number of changed modelling elements to the total number of modelling elements in a model after a delta has been applied.

2.5 Running Example

We present a running example of a car SPL. The core is the software for the basic car model, which has no comfort features. These features are then included in different model variants of the car. We use a visual representation of a derivation tree for to illustrate our running example, as this allows us to demonstrate old and new delta-modelling concepts in Chapter 3. A complete illustration of the running example can be seen in Figure 2.3.

2.5.1 Delta-Modelling

Let $DM = \{\delta_{core}, \delta_{ac}, \delta_{info}, \delta_{window}, \delta_{seat}\}$ be the delta model for our running example, where:

$$\begin{aligned} \delta_{core} &: \text{ adds core,} \\ \delta_{ac} &: \text{ adds air conditioning,} \\ \delta_{info} &: \text{ adds infotainment system,} \\ \delta_{window} &: \text{ adds window heating,} \\ \delta_{seat} &: \text{ adds seat heating.} \end{aligned}$$

We impose following ordering constraints:

- $\delta_{core} \prec \delta_{ac}$ (air conditioning requires core),
- $\delta_{core} \prec \delta_{ac}$ (infotainment system requires core),
- $\delta_{ac} \prec \delta_{window}$ (window heating requires air conditioning),
- $\delta_{ac} \prec \delta_{seat}$ (seat heating requires air conditioning),

In Figure 2.3 we illustrate the derivation tree T of the delta model DM. The root node corresponds to the empty model. The core can than be added by applying the core delta. Each edge represents the application of a single delta. For example, the path

$$(\delta_{core}, \delta_{ac})$$

corresponds to a car equipped with air conditioning. Due to the ordering constraint, δ_{ac} can only be applied after δ_{core}. Nodes corresponding to non-empty sequences represent valid model variants, whereas the root node corresponds to the base configuration and is not considered a model variant. A formal definition of derivation trees can be found in Chapter 3.3.3.1.

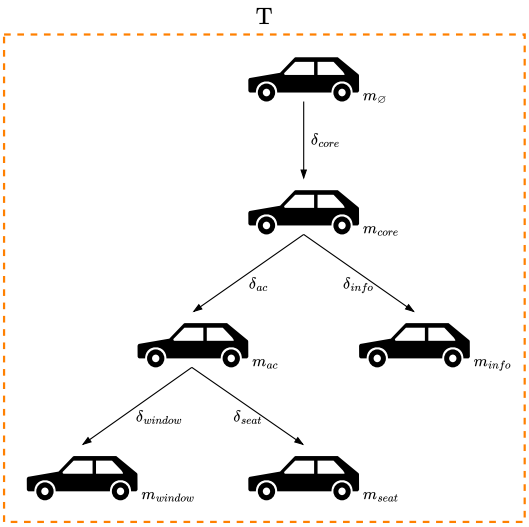
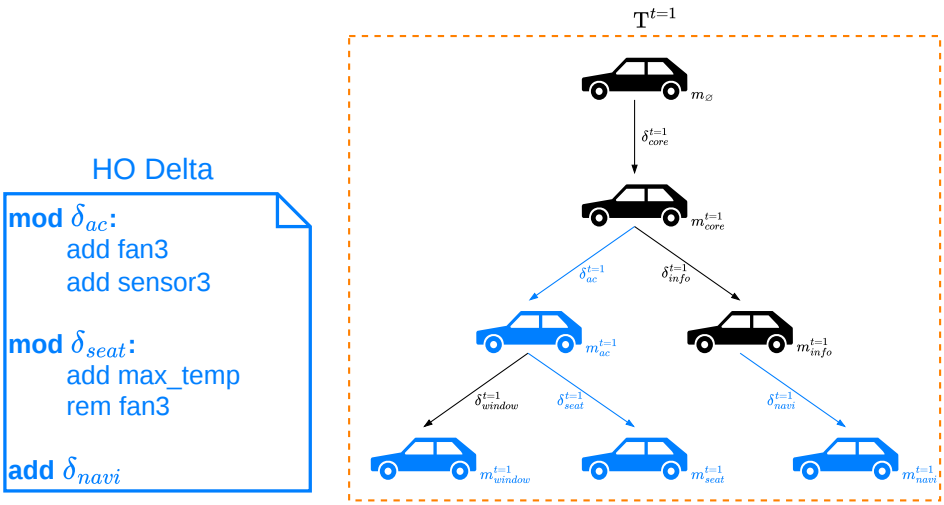


Figure 2.1: Derivation Tree

2.5.2 Higher-Order Delta-Modelling

We evolve our PL by applying the higher-Order delta illustrated in Figure 2.2a on our delta model DM. The higher-Order delta modifies the air conditioning by adding a fan and a sensor, modifies the seat heating by adding a maximum temperature and deleting a fan from the air conditioning, and adds a navigation system. However, the navigation system requires the infotainment system to be added first. Figure 2.2b shows the derivation tree $T^{t=1}$ of the evolved delta model $DM^{t=1}$ can be seen. Each evolved version of a delta, model variant, derivation tree or delta model is marked with a timestamp $^{t=1}$. We have visualised every evolved delta and model variant that has been changed by the higher-Order delta in blue.



(a) HO Delta (b) Evolved Derivation Tree

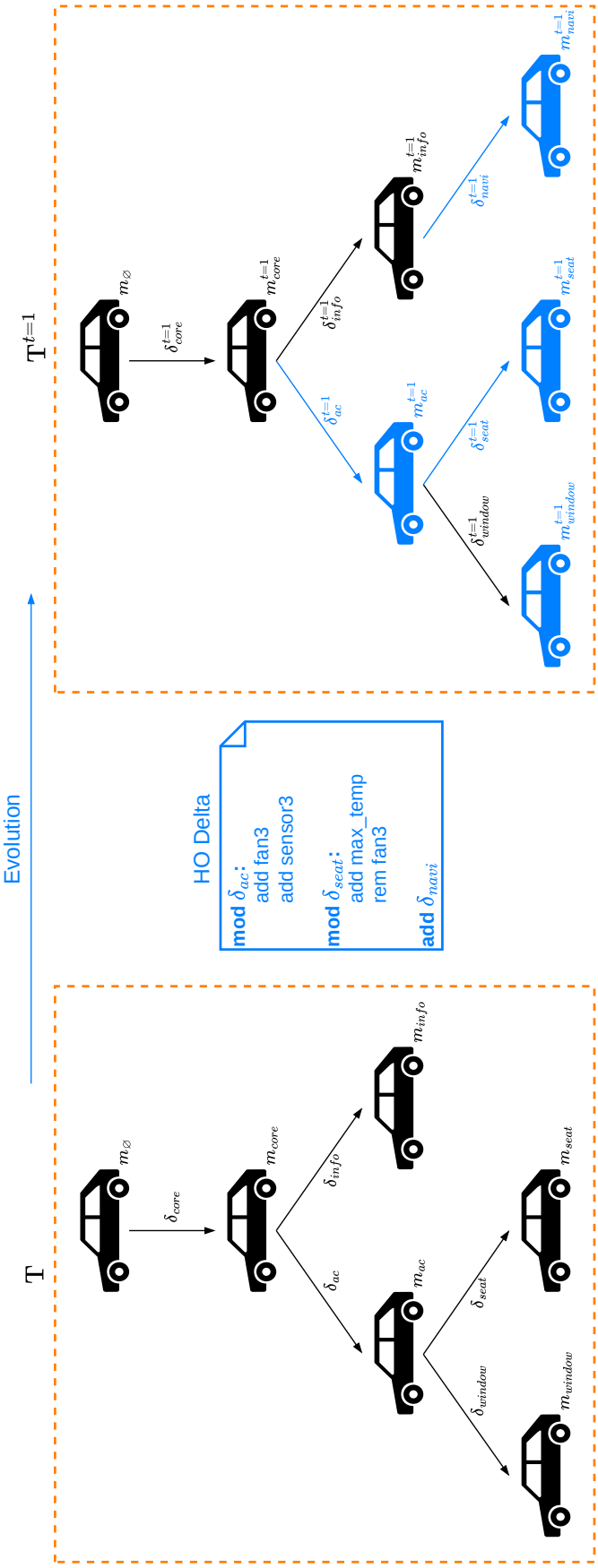


Figure 2.3: Running Example

3. Design

As preliminaries, we introduce the formalisation of Delta-Modelling and higher-order Delta-Modelling in Chapter 3.1. In Chapter 3.2 we propose the concept of the *variant change descriptor*, which describes the model variant evolution with the description model of SGE. In Chapter 3.3 we propose the concept of the *variant space change descriptor*, which describes the evolution of the model variant space. These concepts allow us to estimate the effort and risk introduced by the evolution for each model variant individually as well as for the complete model variant space.

3.1 Preliminaries: Formalizing Delta-Modelling and HO Delta-Modelling

In this chapter, we formally introduce the preliminary concepts. Building on Chapter 2, the formulas and definitions in this chapter are adapted from Lity et. al [10] and aligned with the naming and formalisation conventions of this thesis.

3.1.1 Delta-Modelling

In the following, the delta model $DM = \{\delta_0, ..., \delta_m\} \in \mathcal{DM}$ will be a single-domain delta model, but it will be completely independent of the domain itself. Thus we refer to model variants rather than product variants, as used in Chapter 2.2.

Let $E \subset \mathcal{E}$ be a set of modelling elements. A model m is a tuple $m = (E, R)$ where $R \subseteq E \times E \subset \mathcal{R}$ is a relation over the modelling elements. By $\mathcal{M}$, we refer to the universe of all valid models creatable based on $\mathcal{E}$ and $\mathcal{R}$.

A change operation $op \in \mathcal{R}$ defines an addition/removal/modification of a model element $e \in \mathcal{E}$ (add e/ rem e) or a relation tuple $r \in \nabla$ (add r/ rem r/ mod $(r; r')$), where $\mathcal{OP}$ denotes the universe of all change operations defined over $\mathcal{E}$ and $\mathcal{R}$.

We capture related change operations in basic deltas $\delta = \{op_1, ..., op_l\} \in \mathcal{D}$. By $\mathcal{D} = \mathcal{P}(\mathcal{OP})$, we refer to the universe of all subsets of change operations captured by basic deltas. For transforming a model m_i into model m_j, basic deltas are applied subsequently resulting in a delta sequence $\sigma = (\delta_0, \delta_1, ..., \delta_k) \in \mathcal{D}^*$.

By $\mathcal{DM} = \mathcal{P}(\mathcal{D})$, we refer to the set of all delta models derivable to generate models in $\mathcal{M}$. A delta model has ordering constraints that determine which delta sequences are valid and can therefore be used to generate valid model variants. We define $\mathcal{C} \subseteq \mathcal{D}^*$ as the set of valid delta sequences for our delta model DM.

We define the application function $\text{apply}_\sigma : \mathcal{M} \times \mathcal{D}^* \to \mathcal{M}$ with the empty model $m_\varnothing \in \mathcal{M}$ being the starting point:

- $\text{apply}_\sigma(m, \epsilon) = m$, where ϵ represents the empty delta sequence
- $\text{apply}_\sigma(m, \sigma = (\delta_0, \delta_1, ..., \delta_k)) = \text{apply}_\sigma(apply_\sigma(m, \delta_0), \sigma' = (\delta_1, ..., \delta_k))$
- $\text{apply}_\sigma(m, \delta) = \text{apply}_\sigma(m, \{op_1, op_2, ..., op_l\})$
- $\text{apply}_\sigma(m, \{op_1, op_2, ..., op_l\}) = \text{apply}_\sigma(\text{apply}_\sigma(m, op_1), \{op_2, ..., op_l\})$
- $\text{apply}_\sigma(m, \text{add } e) = m' = (E \cup \{e\}, R)$
- $\text{apply}_\sigma(M, \text{rem } e) = m' = (E \setminus \{e\}, R)$
- $\text{apply}_\sigma(m, \text{add } r(e, e')) = m' = (E, R \cup \{r(e, e')\})$
- $\text{apply}_\sigma(m, \text{rem } r(e, e')) = m' = (E, R \setminus \{r(e, e')\})$
- $\text{apply}_\sigma(m, \text{mod } (r(e, e'), r'(e, e''))) = m' = (E, (R \setminus \{r(e, e')\}) \cup r'(e, e''))$
- $\text{apply}_\sigma(m, \text{mod } (r(e, e'), r'(e'', e'))) = m' = (E, (R \setminus \{r(e, e')\}) \cup r'(e'', e'))$

Each model variant $m_i \in \mathcal{M}$ can now be described as follows: $m_i = m_\varnothing + \sigma_i = m_\varnothing + \delta_1 + ... + \delta_k$ which is equivalent to $m_i = \text{apply}_\sigma(m_\varnothing, \sigma_i) = \text{apply}(\text{apply}(\text{apply}(m_\varnothing, \delta_1), ...), \delta_k)$. We refer to the space containing all model variants that can be generated from the delta model DM as the model variant space $\mathcal{MS}$. Thus, we can define the model variant space as follows:

$$\mathcal{MS} = \{m \in \mathcal{M} \mid \exists \sigma \in \mathcal{C} \ : \text{apply}_\sigma(m_\varnothing, \sigma) = m\}.$$

We define $\cdot : (\mathcal{D}^* \times \mathcal{D}^*) \to \mathcal{D}^*$ as the function that concatenates delta sequences, such that:

$$\sigma_1 \cdot \sigma_2 = \cdot(\sigma_1, \sigma_2) = (\delta_{0_i}, ..., \delta_{k_i}) \cdot (\delta_{0_j}, ..., \delta_{l_j}) = (\delta_{0_i}, ..., \delta_{k_i}, \delta_{0_j}, ..., \delta_{l_j}).$$

Running Example

To generate our model variant m_{seat}, based on our derivation tree illustrated in Figure 2.1, we traverse the tree path from the root node $m_\varnothing$ to m_{seat} and use the path as the delta sequence for our apply function as follows:

$$m_{seat} = \text{apply}_\sigma(m_\varnothing, (\delta_{core}, \delta_{ac}, \delta_{seat})) =$$
$$\text{apply}_\sigma(\text{apply}_\sigma(m_\varnothing, \delta_{core}), (\delta_{ac}, \delta_{seat})) = \text{apply}_\sigma(m_{core}, (\delta_{ac}, \delta_{seat}) =$$
$$\text{apply}_\sigma(\text{apply}_\sigma(m_{core}, \delta_{ac}), \delta_{seat}) = \text{apply}_\sigma(m_{ac}, \delta_{seat})$$

We can also derive the model Variant space $\mathcal{MS}$ from the nodes in the derivation tree, since all of these nodes can be reached using a valid delta sequence:

$$\mathcal{MS} = \{m_\varnothing, m_{core}, m_{core}, m_{ac}, m_{info}, m_{window}, m_{seat}\}$$

3.1.2 Higher-Order Delta-Modelling

After applying the higher-order delta δ^H to the delta model $\mathcal{DM}$ the delta model $\mathcal{DM}$ evolves to $\mathcal{DM}^H = \text{apply}_{HOD}(\mathcal{DM}, \delta^H)$. We define the application function $\text{apply}_{HOD} : \mathcal{DM} \times \mathcal{P}(\mathcal{OP}^H) \rightarrow \mathcal{DM}$ as follows:

- $\text{apply}_{HOD}(DM, \emptyset) = DM$
- $\text{apply}_{HOD}(DM, \delta^H) = \text{apply}_{HOD}(DM, \{op_1^H, op_2^H, ..., op_l^H\})$
- $\text{apply}_{HOD}(DM, \{op_1^H, op_2^H, ..., op_l^H\}) = \text{apply}_{HOD}(\text{apply}_{HOD}(DM, op_1^H), \{op_2^H, ..., op_l^H\})$
- $\text{apply}_{HOD}(DM, \text{add } \delta) = DM \cup \{\delta\}$
- $\text{apply}_{HOD}(DM, \text{rem } \delta) = DM \setminus \{\delta\}$
- $\text{apply}_{HOD}(DM, \text{mod}(\delta, \text{add}op, ..., \text{rem } op', ...)) = (DM \setminus \{\delta\}) \cup \{\delta\}$, where $\delta' = (\delta \cup \{op, ...\})\{op', ...\}$

To distinguish between different versions of model variants, we assign a timestamp to each one to represent the evolutionary steps. For example, a model variant $m_i^{t=j}$ would evolve into $m_i^{t=j+1}$ after one higher-order delta has been applied to the delta model. We refer to $m_i^{t=j+1}$ as an evolved model variant. If no timestamp is shown, t is zero. Evolved deltas, delta sequences and delta models are also superscripted with a timestamp.

Running Example

The evolved delta model $DM^{t=1}$ is obtained by applying the higher-order delta δ^H illustrated in Figure 2.2a:

$$DM^{t=1} = \text{apply}_{HOD}(DM, \delta^H) =$$
$$\text{apply}_{HOD}(DM, \{\text{mod } \delta_{ac}, \text{mod } \delta_{seat}, \text{add } \delta_{navi}\}) =$$
$$\text{apply}_{HOD}(\text{apply}_{HOD}(\text{apply}_{HOD}(DM, \text{mod } \delta_{ac}), \text{mod } \delta_{seat}), \text{add } \delta_{navi})) =$$
$$\{\delta_{core}^{t=1}, \delta_{ac}^{t=1}, \delta_{info}^{t=1}, \delta_{window}^{t=1}, \delta_{seat}^{t=1}, \delta_{navi}^{t=1}\}$$

The deltas $\delta_{core}^{t=1}$, $\delta_{info}^{t=1}$, and $\delta_{window}^{t=1}$ are unchanged by the higher-order delta. In contrast, operations were added to δ_{ac} and δ_{seat}. More precisely, $\delta_{ac}^{t=1}$ contains the operations of δ_{ac} plus $\{\text{add } fan3, \text{add } sensor3\}$, and $\delta_{seat}^{t=1}$ contains the operations of δ_{seat} plus $\{\text{add } max\_temp, \text{rem } fan3\}$. Finally, $\delta_{navi}^{t=1}$ is newly added by the higher-order delta. Before the evolution, no corresponding base delta exists for this newly introduced functionality, thus δ_{navi} is an empty set.

3.2 Describing Evolution of Individual Model Variants

This chapter introduces the *variant change descriptor*. Its purpose is to describe the evolution of a single model variant when a higher-order delta is applied to the delta model. This descriptor allows the effort and risk introduced by an evolution to be estimated for each affected model variant individually.

The calculation consists of two steps. First, the concrete change of a model variant is derived by comparing the delta sequence that generates the model variant before the evolution with the delta sequence that generates the corresponding model variant after the evolution. This comparison is performed by a diffing function. Second, the resulting change is mapped to the description model of SGE. The result of this mapping is the *variant change descriptor*. Since delta sequences may contain operations that cancel or overwrite each other, a reduction function is introduced before the descriptor is calculated. The reduction function removes redundant delta operations so that the descriptor counts only those operations that actually contribute to the evolved model variant.

Chapter 3.2.1 introduces the diffing function and Chapter 3.2.2 the projected higher-order delta. Both concepts are combined for defining the evolution of an individual model variant in Chapter 3.2.3. Chapter 3.2.4 then introduces the reducible function, enabling the *variant change descriptor* to be calculated in Chapter 3.2.5.

3.2.1 Delta Diffing

The diffing function compares two delta sequences: one sequence that generates the base model variant and one sequence that generates the evolved model variant. The output states which deltas and operations are inserted, deleted, or modified. This information is later used to determine whether a higher-order delta affects a specific model variant and, if so, which change operations have to be considered for the variant change descriptor.

Because the two delta sequences may have different lengths, they cannot always be compared position by position. For example, an evolved sequence may contain an additional delta introduced by the higher-order delta. Therefore, an *alignment* is used first. The alignment matches corresponding deltas in both sequences while preserving their order. Deltas that do not have a counterpart are aligned with the gap symbol $\perp$. From this alignment, two padded sequences are obtained. These padded sequences have the same length, so that the diff can compare them element by element.

Pointwise diff of delta sequences with alignment.

Let $\mathcal{D}^*$ be the universe of all subsets of deltas and let $\perp \notin \mathcal{D}^*$ be a distinguished gap symbol. Define the extended universe $\mathcal{D}^*_\perp := \mathcal{D}^* \cup \{\perp\}$.

Let

$$\sigma_1 = (\delta_1^{(1)}, \ldots, \delta_n^{(1)}) \in \mathcal{D}^*, \quad \sigma_2 = (\delta_1^{(2)}, \ldots, \delta_m^{(2)}) \in \mathcal{D}^*$$

be two delta sequences.

An alignment $A \subseteq \{1, \ldots, n\} \times \{1, \ldots, m\}$ is an order-preserving partial bijection. It specifies which deltas in σ_1 correspond to which deltas in σ_2. If a delta has no

corresponding partner, it is paired with $\perp$ in the padded sequence. The alignment$A$ induces padded sequences

$$\tilde{\sigma}_1 = (\tilde{\delta}_1^{(1)}, \ldots, \tilde{\delta}_k^{(1)}), \quad \tilde{\sigma}_2 = (\tilde{\delta}_1^{(2)}, \ldots, \tilde{\delta}_k^{(2)})$$

with $\tilde{\delta}_l^{(1)}, \tilde{\delta}_l^{(2)} \in \mathcal{D}_\perp^*$. Removing all occurrences of $\perp$ from $\tilde{\sigma}_1$ yields σ_1, and removing all occurrences of $\perp$ from $\tilde{\sigma}_2$ yields σ_2.

The pointwise diff induced by A is then defined as

$$\mathrm{diff}_\sigma^A(\sigma_1, \sigma_2) := (\tilde{\delta}_1^{(1)} \ominus \tilde{\delta}_1^{(2)}, \ldots, \tilde{\delta}_k^{(1)} \ominus \tilde{\delta}_k^{(2)}) \in \mathcal{D}^*.$$

Each pair of aligned deltas is compared by the delta difference operator $\ominus$.

Delta difference operator.

Let $\perp \notin D$ be a gap symbol and define $\mathcal{D}_\perp = \mathcal{D} \cup \{\perp\}$. The delta difference operator

$$\ominus : \mathcal{D}_\perp \times \mathcal{D}_\perp \to \mathcal{D}$$

returns a diff atom for two aligned deltas. If both deltas are equal, the result records equality. If both deltas exist but differ, their contained operations are compared by an operation-level diff. If one side is a gap, the delta is treated as inserted or deleted:

$$\delta \ominus \delta' := \begin{cases} \mathrm{eq}(\delta), & \text{if } \delta = \delta', \\ \mathrm{diff}_{op}^B(\delta, \delta'), & \text{if } \delta \neq \delta', \\ \mathrm{del}(\delta), & \text{if } \delta' = \perp, \\ \mathrm{ins}(\delta'), & \text{if } \delta = \perp. \end{cases}$$

Thus, the delta difference operator lifts the comparison from whole delta sequences to the level of individual deltas.

Alignment-based diff on delta operations.

If two aligned deltas differ, their operations are compared analogously. Let B be an order-preserving partial bijection between the operations of the two deltas. The operation-level diff induced by B is defined as

$$\mathrm{diff}_{op}^B(\delta, \delta') := (\tilde{o}_1 \ominus_{op} \tilde{o}_1', \ldots, \tilde{o}_k \ominus_{op} \tilde{o}_k') \in \mathcal{D}.$$

With $\mathcal{OP}_\perp = \mathcal{OP} \cup \{\perp\}$, the operation difference operator is defined by

$$o \ominus_{op} o' := \begin{cases} \mathrm{eq}(o), & \text{if } o = o', \\ \mathrm{sub}(o, o'), & \text{if } o \neq o' \text{ and } o, o' \in \mathcal{OP}, \\ \mathrm{del}(o), & \text{if } o' = \perp, \\ \mathrm{ins}(o'), & \text{if } o = \perp. \end{cases}$$

The result identifies which operations were preserved, replaced, removed, or inserted. Only the inserted, deleted, and substituted operations are relevant for the concrete change induced by the evolution.

Running Example

The question is which operations were introduced or removed on the path that generates the evolved model variant $m_{seat}^{t=1}$. To answer this, the base sequence σ_{seat} is compared with the evolved sequence $\sigma_{seat}^{t=1}$:

$$\begin{aligned}\mathrm{diff}_{\sigma}^{A}(\sigma_{seat}, \sigma_{seat}^{t=1}) &= (\tilde{\delta}_{core} \ominus \tilde{\delta}_{core}^{t=1}, \tilde{\delta}_{ac} \ominus \tilde{\delta}_{ac}^{t=1}, \tilde{\delta}_{seat} \ominus \tilde{\delta}_{seat}^{t=1}) \\ &= (\mathrm{diff}_{op}^{B}(\tilde{\delta}_{ac}, \tilde{\delta}_{ac}^{t=1}), \mathrm{diff}_{op}^{B}(\tilde{\delta}_{seat}, \tilde{\delta}_{seat}^{t=1})) \\ &= (\{\text{add } fan3, \text{add } sensor3\}, \{\text{add } max\_temp, \text{rem } fan3\}).\end{aligned}$$

The result shows that the evolved path to $m_{seat}^{t=1}$ contains four explicitly changed operations: two operations added to δ_{ac} and two operations added to δ_{seat}. These operations are the explicit changes introduced by the higher-order delta on the derivation path of m_{seat}.

3.2.2 Projected Higher-Order Delta

Let $\mathrm{p}_{HOD} : \mathcal{M} \times \mathcal{P}(OP^H) \to \mathcal{P}(OP^H)$ be the projection function. It is defined recursively as follows:

- $\mathrm{p}_{HOD}(m, \emptyset) = \emptyset$,
- $\mathrm{p}_{HOD}(m, \delta^H) = \mathrm{p}_{HOD}(m, \{op_1^H, op_2^H, \ldots, op_l^H\})$,
- $\mathrm{p}_{HOD}(m, \{op_1^H, op_2^H, \ldots, op_l^H\}) = \mathrm{p}_{HOD}(m, op_1^H) \cup \cdots \cup \mathrm{p}_{HOD}(m, op_l^H)$.

For a single higher-order operation op^H, the projection contains op^H if the delta changed by op^H occurs in a valid delta sequence that generates m. Otherwise, the operation is ignored:

$$\mathrm{p}_{HOD}(m, op^H) = \begin{cases} op^H, & \text{if } \{\exists \sigma \in DM^* : apply_{\sigma}(m_{\varnothing}, \sigma) = m \mid \mathrm{diff}_{\sigma}^{A}(\sigma^{t=0}, \sigma^{t=1}) \neq \emptyset\} \\ \emptyset, & \text{otherwise.} \end{cases}$$

The projection therefore filters the higher-order delta by relevance for one model variant. The filtered result can be used to explain why the selected model variant changed.

Running Example

For the model variant m_{seat}, the relevant derivation path contains δ_{core}, δ_{ac}, and δ_{seat}. The higher-order delta changes δ_{ac}, δ_{seat}, and adds δ_{navi}. Since δ_{navi} is not part of the path to m_{seat}, it is not included in the projection:

$$\mathrm{p}_{HOD}(m_{seat}, \delta^H) = (\text{mod } \delta_{ac} : \{\text{add } fan3, \text{add } sensor3\},$$

$$\text{mod } \delta_{seat} : \{\text{add } max\_temp, \text{rem } fan3\})$$

This result identifies the higher-order operations that explain the evolution of m_{seat}.

3.2.3 Individual Model Variant Evolution

The evolution of an individual model variant is described in two complementary ways. The projected higher-order delta explains which higher-order operations are relevant for the model variant. The change delta describes the concrete operation-level difference between the base and evolved derivation sequences.

Let $DM = \{\delta_0, \ldots, \delta_m\} \in \mathcal{DM}$ be a delta model, let δ^H be a higher-order delta, and let m_i be a valid model variant in $\mathcal{MS}$. The evolution relevant to m_i is defined as

$$evo_{m_i} := \mathrm{p}_{HOD}(m_i, \delta^H).$$

For the concrete change, let $\sigma_i^{t=0}$ be a valid delta sequence that generates $m_i^{t=0}$:

$$apply_\sigma(m_\emptyset, \sigma_i^{t=0}) = m_i^{t=0}.$$

Let $\sigma_i^{t=1}$ be a valid delta sequence that generates the corresponding evolved model variant $m_i^{t=1}$:

$$apply_\sigma(m_\emptyset, \sigma_i^{t=1}) = m_i^{t=1}.$$

If the evolved model variant did not exist before the evolution, the base sequence is chosen as the sequence of the closest predecessor model variant in the evolved derivation tree. The closest predecessor is the model variant with the shortest path to the newly introduced variant.

With an alignment A between $\sigma_i^{t=0}$ and $\sigma_i^{t=1}$, the concrete change of m_i is defined as

$$chg_{m_i} := \mathrm{diff}_\sigma^A(\sigma_i^{t=0}, \sigma_i^{t=1}).$$

The result chg_{m_i} is a delta sequence-level diff. For the descriptor calculation, the changed operations contained in this diff are collected in a single change delta, denoted by $\delta_{chg_{m_i}}$.

Running Example

For m_{seat}, the relevant higher-order evolution and the concrete change are

$$evo_{m_{seat}} = \mathrm{p}_{HOD}(m_{seat}, \delta^H)$$

and

$$chg_{m_{seat}} = \mathrm{diff}_\sigma^A(\sigma_{seat}^{t=0}, \sigma_{seat}^{t=1}).$$

The projection explains which higher-order operations are responsible for the evolution, while the diff produces the concrete changed operations. In the running example, the collected change delta is

$$\delta_{chg_{m_{seat}}} = \{\text{add } fan3, \text{add } sensor3, \text{add } max\_temp, \text{rem } fan3\}.$$

3.2.4 Reducibility of a Delta

Before the *variant change descriptor* is calculated, *redundant* operations, which are operations that either cancel or overwrite each other, have to be removed from the change delta and from the evolved delta sequence. This is necessary because the

descriptor measures the share of changed modelling elements. If operations that cancel or overwrite each other are counted, the descriptor overestimates the actual change. For example, adding an element and removing the same element again does not change the final model variant, even though two operations occur in the delta sequence.

The reduction is applied to operation sequences, not to model variants themselves. A model variant is generated by applying a delta operation sequence. If one or more operations can be deleted from that sequence without changing the generated model variant, the sequence is *reducible*. A higher-order delta preserves non-reducibility if the evolved delta operation sequences remain *irreducible* after the evolution. This condition is relevant because an evolution can introduce new operations that make formerly *irreducible* sequences reducible.

If a delta model is increasing monotonic, it only contains add operations and mod operations that only extends existing elements and does not change the already existing body of the element, as explained in Chapter 2.2.1. Therefore, if every valid delta sequence contained in a delta model is *irreducible*, then the delta model is increasing monotonic.

Reducibility

A delta

$$\delta = (op_1, \ldots, op_l) \in \mathcal{D}$$

is called *reducible* if there exists a proper subsequence $\delta' \sqsubset \delta$ such that applying δ' has the same effect as applying δ. Otherwise, δ is called *irreducible*.

Reduction Function

A *reduction function* is a function

$$\text{reduce} : \mathcal{D} \to \mathcal{D}$$

such that, for every delta $\delta \in \mathcal{D}$, the following properties hold:

1. **Semantic preservation:** reduce(δ) has the same effect as δ.
2. **Subsequence property:** reduce(δ) $\sqsubseteq \delta$.
3. **Irreducibility:** reduce(δ) is irreducible.

Thus, the reduction function removes only operations that are redundant for the final generated model.

Reduction Rules

Let $\mathcal{T}$ be the universe of model artefacts, consisting of modelling elements and relations. For every delta operation $op \in \mathcal{OP}$, let

$$\text{target}(op) \in \mathcal{T}$$

denote the artefact affected by op. Furthermore, let scope(op) denote the part of the artefact that is modified by op. For modification operations, the scope may for example be an attribute, a reference, or the body of an element.

For a delta $\delta = (op_1, \ldots, op_l) \in \mathcal{D}$, a set of operations is redundant if it satisfies one of the following conditions.

1. **Add–remove elimination.** There exist indices $i < j$ and an artefact $x \in \mathcal{T}$ such that $op_i = \text{add}(x)$ and $op_j = \text{rem}(x)$. If every operation between op_i and op_j that targets x becomes undefined once $\text{add}(x)$ and $\text{rem}(x)$ are removed, then the complete set

$$\{op_i, op_j\} \cup \{op_k \mid i < k < j, \text{target}(op_k) = x\}$$

may be deleted.

2. **Overwritten modification elimination.** There exist indices $i < j$ and an artefact $x \in \mathcal{T}$ such that $op_i = \text{mod}(x, s, v)$ and $op_j = \text{mod}(x, s, v')$. If both operations modify the same scope s of the same artefact x and no operation between them modifies the same scope of x, then op_i may be deleted. The later modification overwrites the earlier one.

Running Example

For m_{seat}, the changed operations are collected in the change delta

$$\delta_{chg_{m_{seat}}} = \{\text{add } fan3, \text{add } sensor3, \text{add } max\_temp, \text{rem } fan3\}.$$

The operations add $fan3$ and rem $fan3$ cancel each other. Therefore, they are removed by the reduction function:

$$\text{reduce}(\delta_{chg_{m_{seat}}}) = \{\text{add } sensor3, \text{add } max\_temp\}.$$

The result shows that the final evolved model variant differs from the base variant by two effective operations, not by four explicitly introduced operations. This distinction is important for the descriptor calculation in the next section.

3.2.5 Variant Change Descriptor

The variant change descriptor describes the evolution of a single model variant using the description model of SGE. It consists of three parts:

1. the concrete change delta $\delta_{chg_{m_i}}$,
2. the share of changed modelling elements $SoCME_{m_i}$,
3. the distribution of the changed operations across the SGE variation classes used in this thesis.

The descriptor therefore states not only which operations changed, but also how large the effective change is in relation to the evolved model variant and which kind of impact the change has.

To classify changed operations, we use the composed mapping from deltas to the description model of SGE based on the preliminary work of Ochs et al. [15]. In the application, this composed mapping generates an aggregated change descriptor. For the descriptor introduced here, the relevant part of this mapping is represented as

$$mapping : \mathcal{OP} \to \{AV, PV\}.$$

An AV denotes a change of characteristics or attributes of a model. A PV denotes a change of the functioning principle of a model.

For a change delta $\delta_{chg_{m_i}}$, the changed operations are divided into

$$\delta^{AV}_{chg_{m_i}} = \{op \in \delta_{chg_{m_i}} \mid mapping(op) = AV\}$$

and

$$\delta^{PV}_{chg_{m_i}} = \{op \in \delta_{chg_{m_i}} \mid mapping(op) = PV\}.$$

The relative distribution inside the changed operations is defined as

$$AV_{m_i} := \frac{|\delta^{AV}_{chg_{m_i}}|}{|\delta_{chg_{m_i}}|}$$

and

$$PV_{m_i} := \frac{|\delta^{PV}_{chg_{m_i}}|}{|\delta_{chg_{m_i}}|}.$$

The share of changed modelling elements compares the effective change with the size of the evolved model variant. In this thesis, modelling elements are represented by the delta operations that generate the model variant. Let $\delta^{t=1}_{m_i}$ be the concatenation of all deltas in the evolved delta sequence $\sigma^{t=1}_i$. The number of modelling elements of $m^{t=1}_i$ is

$$el_{m_i} = |\delta^{t=1}_{m_i}| = \sum_{\delta^{t=1}_j \in \sigma^{t=1}_i} |\delta^{t=1}_j|.$$

The share of changed modelling elements is then defined as

$$SoCME_{m_i} := \frac{|\delta_{chg_{m_i}}|}{el_{m_i}}.$$

For a correct calculation, both the change delta $\delta_{chg_{m_i}}$ and the evolved delta sequence used to calculate el_{m_i} must be reduced first. Otherwise, redundant operations are counted as changed modelling elements, even though they do not affect the final generated model variant. Thus, the corrected calculation is

$$SoCME^{reduced}_{m_i} := \frac{|reduce(\delta_{chg_{m_i}})|}{|reduce(\delta^{t=1}_{m_i})|}.$$

The AV and PV shares are calculated on the reduced change delta as well.

Running Example

Assume the following numbers of operations for the deltas in the running example:

$$\begin{aligned} |\delta_{core}| &= 20, \\ |\delta_{ac}| &= 16, \\ |\delta_{info}| &= 25, \\ |\delta_{window}| &= 8, \\ |\delta_{seat}| &= 10. \\ |\delta_{navi}^{t=1}| &= 20. \end{aligned}$$

For m_{seat}, the unreduced change delta contains four operations:

$$\delta_{chg_{m_{seat}}} = \{\text{add } fan3, \text{add } sensor3, \text{add } max\_temp, \text{rem } fan3\}.$$

Without reduction, the share of changed modelling elements is

$$SoCME_{m_{seat}} = \frac{4}{50}.$$

After reduction, the effective change delta contains only two operations:

$$reduce(\delta_{chg_{m_{seat}}}) = \{\text{add } sensor3, \text{add } max\_temp\}.$$

The evolved operation sequence used to generate $m_{seat}^{t=1}$ is also reduced, which changes the denominator from 50 to 48. Thus,

$$SoCME_{m_{seat}}^{reduced} = \frac{2}{48}.$$

The comparison shows that the unreduced calculation overestimates the effective change. It counts operations that occur syntactically in the evolution but cancel each other before the final evolved model variant is obtained. The reduced descriptor therefore better represents the actual effort and risk for the model variant.
If, for example, both effective operations add $sensor3$ and add max_temp are classified as AVs, then

$$AV_{m_{seat}} = 1, \qquad PV_{m_{seat}} = 0.$$

If one of the two operations is classified as a PV, then both values would be $\frac{1}{2}$. The exact values therefore depend on the SGE mapping used for the concrete modelling domain.

3.3 Describing Evolution of the Model Variant Space

The previous section described the evolution of individual model variants (SG1). This section describes the evolution of the complete model variant space. The corresponding descriptor is called the *variant space change descriptor*. It states which share of the modelling elements across all model variants has changed and how the changed operations are distributed across the SGE variation classes. Thereby we can estimate the effort and risk introduced by the evolution.

Two calculation paths are introduced, illustrated in Figure 3.1. The green path (SG2a) aggregates the variant change descriptors of all model variants. This path is straightforward and traceable, but it requires the explicit calculation of every individual variant change descriptor. For delta models with many variants, this can become expensive. The orange path (SG2b) calculates the variant space change descriptor directly from the evolved derivation tree and the changed deltas. This avoids calculating a complete descriptor for every model variant and is therefore intended as a more scalable alternative.

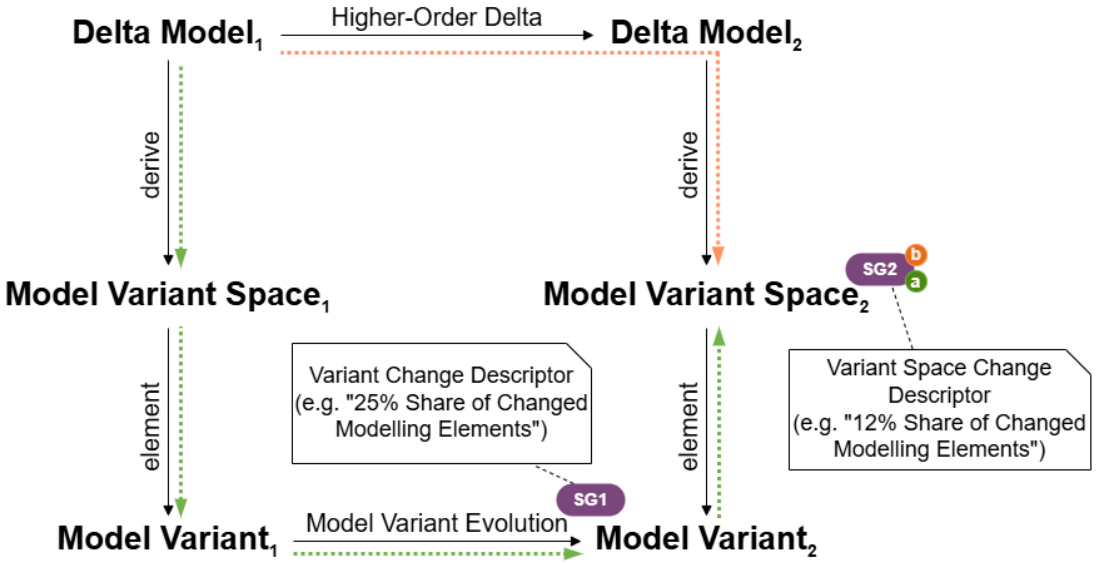


Figure 3.1: Sub-Goals

Both paths use the 150% model introduced in Chapter 3.3.1, to calculate how much of the model variant space has changed. Chapter 3.3.2 presents the aggregation-based path. Chapter 3.3.3 presents the direct path.

3.3.1 150% Model

The *variant space change descriptor* must account for the fact that the same delta operation can affect several model variants. Therefore, the descriptor does not count each operation only once at the delta-model level. Instead, it counts the operation once for every model variant in which it is effective. This corresponds to the idea of a 150% model in PL engineering: the 150% model represents the modelling elements of all model variants together [17].

In this thesis, the 150% model is used as the denominator for the *variant space change descriptor*. It contains the modelling elements of all evolved model variants, where modelling elements are represented by the reduced delta operation sequences that generate these variants. This definition is necessary because redundant operations may be effective in one variant but redundant in another, depending on the complete derivation path.

For each model variant $m_i \in MS$, let el_{m_i} be the number of reduced operations in the evolved delta sequence that generates $m_i^{t=1}$. The size of the 150% model is defined as

$$el_{150\%} := \sum_{m_i \in MS} el_{m_i}.$$

This value is then used to normalise the aggregated changes across the entire model variant space.

Running Example

For the evolved delta model $DM^{t=1}$, the 150% model is obtained by summing the modelling elements of all evolved model variants. Without reduction of the modelling elements, the example yields

$$\begin{aligned} el_{150\%} &= el_{m_{core}} + el_{m_{ac}} + el_{m_{info}} + el_{m_{window}} + el_{m_{seat}} + el_{m_{navi}} \\ &= 20 + 38 + 45 + 46 + 50 + 65 = 218. \end{aligned}$$

After reduction, redundant operations on the path to m_{seat} are removed. Therefore,

$$el_{150\%}^{reduced} = 20 + 38 + 45 + 46 + 48 + 65 = 216.$$

The difference shows that reduction affects the size of the 150% model because the 150% model is calculated over the effective operation sequences of all model variants.

3.3.2 Aggregating Variant Change Descriptors

The *variant space change descriptor* is the share of changed modelling elements in the model variant space, as well as the AV and PV contributions. The aggregation-based path calculates the *variant space change descriptor* from the *variant change descriptor* of all model variants. For each model variant, the individual descriptor provides the changed deltas and the number of modelling elements in the evolved model variant. These values are then aggregated over the complete model variant space and normalised by the size of the 150% model.

Let $SoCME_{\mathcal{MS}}$ be the share of changed modelling elements in our model variant space $\mathcal{MS}$, where modelling elements are defined as delta operations. We can then define $SoCME_{\mathcal{MS}}$ as follows:

$$SoCME_{\mathcal{MS}} := \sum_{m_i \in \mathcal{MS}} \left(SoCME_{m_i} \times \frac{el_{m_i}}{el_{150\%}} \right) = \frac{\sum_{m_i \in \mathcal{MS}} |\delta_{chg_{m_i}}|}{el_{150\%}}$$

This formulation shows that each model variant contributes according to its size. A change in a larger model variant therefore has a larger impact on the overall descriptor than an equally sized change in a smaller model variant.

The AV and PV contributions are aggregated analogously:

$$SoCME^{AV}_{\mathcal{MS}} := \frac{\sum_{m_i \in \mathcal{MS}} |\delta^{AV}_{chg_{m_i}}|}{el_{150\%}}.$$

Analogously, the aggregated PV share is defined as

$$SoCME^{PV}_{\mathcal{MS}} := \frac{\sum_{m_i \in \mathcal{MS}} |\delta^{PV}_{chg_{m_i}}|}{el_{150\%}}.$$

Together, these two values decompose the overall variant space change descriptor:

$$SoCME_{\mathcal{MS}} = SoCME^{AV}_{\mathcal{MS}} + SoCME^{PV}_{\mathcal{MS}}.$$

While $SoCME^{AV}_{\mathcal{MS}}$ and $SoCME^{PV}_{\mathcal{MS}}$ describe the contribution of AVs and PVs relative to all modelling elements in the 150% model, we can additionally express the distribution within the changed modelling elements themselves. The relative AV and PV shares of the aggregated change are defined as

$$AV_{\mathcal{MS}} := \frac{\sum_{m_i \in \mathcal{MS}} |\delta^{AV}_{chg_{m_i}}|}{\sum_{m_i \in \mathcal{MS}} |\delta_{chg_{m_i}}|}$$

and

$$PV_{\mathcal{MS}} := \frac{\sum_{m_i \in \mathcal{MS}} |\delta^{PV}_{chg_{m_i}}|}{\sum_{m_i \in \mathcal{MS}} |\delta_{chg_{m_i}}|}.$$

Thus, the aggregated descriptor states both how large the change is in relation to the whole model variant space and how this change is distributed between AVs and PVs.

Running Example

In the running example, only the variants with non-empty change deltas are included in the numerator. Without reduction, the changed operations are counted as follows:

$$SoCME_{MS} = \frac{|\delta_{chg_{m_{ac}}}| + |\delta_{chg_{m_{window}}}| + |\delta_{chg_{m_{seat}}}| + |\delta_{chg_{m_{navi}}}|}{el_{150\%}} = \frac{2+2+4+20}{218} = \frac{28}{218}.$$

After reduction, the redundant operations in the change of m_{seat} are removed, and the denominator is adjusted to the reduced 150% model:

$$SoCME_{MS}^{reduced} = \frac{2+2+2+20}{216} = \frac{26}{216}.$$

The comparison shows that the unreduced calculation overestimates the effective change in the model variant space. The value 28/218 counts operations that are introduced by the higher-order delta but cancel each other in at least one generated model variant. The reduced value 26/216 counts only operations that remain effective in the generated variants. Therefore, the reduced calculation gives a more accurate estimate of the effort and risk introduced by the evolution in the running example.

3.3.3 Variant Space Change Descriptor

We calculate the *variant space change descriptor* directly, using only the deltas, higher-order delta and ordering constraints. The *variant space change descriptor*is the share of changed modelling elements in the model variant space, as well as the AV and PV contributions. These calculations are based on the orange path for SG2b, illustrated in Figure 3.1. Before we begin with the calculations in Chapter 3.3.3.2, we formally introduce the delta derivation tree in the following chapter, since it is used as a traversing structure for the direct calculations.

The direct calculation path computes the variant space change descriptor without first constructing the variant change descriptor for every model variant. Instead, it uses the evolved derivation tree to identify subtrees in which the same changed delta operations affect all contained model variants. Each subtree can then be counted once and weighted by the number of valid model variants it contains.

This path uses the same descriptor values as the aggregation-based path: $SoCME_{MS}$, $SoCME_{MS}^{AV}$, $SoCME_{MS}^{PV}$, AV_{MS}, and PV_{MS}. The difference lies only in the way the numerator is calculated. Before we begin with the calculations in Chapter 3.3.3.2, we formally introduce the delta derivation tree in the following chapter, since it is used as a traversing structure for the direct calculations.

3.3.3.1 Delta Derivation Tree

The direct calculation uses a delta derivation tree as a traversal structure. A delta derivation tree represents derivation paths through a delta model. Edges are labelled with deltas, and each node stores the delta sequence from the root to that node. If the sequence is valid, the node also represents the model variant generated by that sequence.

A *delta derivation tree* is a rooted tree (V, E, v_0) together with

- an edge-labelling function $\lambda : E \to \mathcal{D}$,
- a sequence assignment $\sigma : V \to \mathcal{D}^*$,
- a partial model assignment $\mu : V \rightharpoonup \mathcal{M}$,

such that

$$\sigma(v_0) = \emptyset, \qquad \mu(v_0) = m_\varnothing,$$

and for every edge $(u, v) \in E$,

$$\sigma(v) = \sigma(u) \cdot \lambda(u, v).$$

Moreover, for every node $v \in V$, $\sigma(v) \in \mathcal{D}^*$. Whenever $\sigma(v) \in \mathcal{C}$, is true, then $\mu(v)$ is defined as

$$\mu(v) = \text{apply}_\sigma(m_\varnothing, \sigma(v)).$$

The direct calculation splits this derivation tree whenever a changed delta becomes effective for the following subtree. The split condition is therefore not an arbitrary attribute. It is the condition that the edge leading to a node contains changed operations that are effective after reduction. The resulting split-induced subtree forest groups model variants by the changed operations they inherit from their subtree root.

Let

$$T = (V, E, \lambda, \sigma, \mu, v_0)$$

be a delta derivation tree, and let

$$\mathit{SplitCondition} : V \times D \to \{\mathit{true}, \mathit{false}\}$$

be the predicate that is true if the delta on the incoming path introduces an effective change. A split-induced subtree forest of T is a set

$$F = \{T_1, \ldots, T_k\}$$

of rooted delta derivation trees such that the root of T_1 is v_0, every node and edge of T belongs to exactly one subtree, and every node satisfying the split condition is the root of a subtree. Thus, the trees in F partition the original derivation tree into regions with the same inherited change.

Running Example

Figure 2.1 provides a visual representation of the derivation tree for our running example. The annotated edges show the results of the edge-labelling function λ. In this example, each node is also a model variant, thus showing the model assignment μ. Each edge sequence leading to a node is a valid delta sequence for the model variant and therefore shows a valid sequence assignment σ. An example of a split-induced forest tree can be found in the running example in Chapter 3.3.3.

3.3.3.2 Direct Calculation Path

For a delta model $DM = \{\delta_0, ..., \delta_m\} \in \mathcal{DM}$, let δ^H be a higher-order delta such that $DM^{t=1} = apply_{HOD}(DM, \delta^H)$. Let T be the derivation tree of DM and $T^{t=1}$ of $DM^{t=1}$. For every delta δ_i, the changed operations are calculated by comparing the base and evolved versions of that delta:

$$chg_{\delta_i} := \text{diff}^B_{op}(\delta_i^{t=0}, \delta_i^{t=1}).$$

In addition to the set of changed delta operations, each changed operation is classified according to the AV/PV mapping introduced for the *variant change descriptor*. For a change delta chg_{δ_i}, we define

$$chg^{AV}_{\delta_i} = \{op \in chg_{\delta_i} \mid mapping(op) = AV\}$$

and

$$chg^{PV}_{\delta_i} = \{op \in chg_{\delta_i} \mid mapping(op) = PV\}.$$

Thus, the changed operations of a delta can be decomposed as

$$|chg_{\delta_i}| = |chg^{AV}_{\delta_i}| + |chg^{PV}_{\delta_i}|.$$

Algorithm 1 traverses the evolved derivation tree depth-first. Whenever the algorithm reaches a model variant whose incoming path contains effective changed operations, it starts a new subtree. The root of this subtree stores the reduced change that affects all model variants inside the subtree. The helper function HasChanged identifies whether a changed delta is present. The helper function ReduceChange removes changed operations that become redundant on the complete path to the current model variant. In this way, only effective changes are counted.

After the tree has been split, Algorithm 4 counts the valid model variants in each subtree. Algorithm 5 then multiplies this number by the changed operations stored at the root of the subtree. This works because every valid model variant in the subtree inherits the effective change stored at the subtree root.

Therefore, we apply Algorithm 1 to $T^{t=1}$ to obtain the set of subtrees $\mathcal{F}$ and then apply Algorithm 5 to $\mathcal{F}$ to obtain $el_{\mathcal{F}}$, $el^{AV}_{\mathcal{F}}$, and $el^{PV}_{\mathcal{F}}$. We can then calculate the *variant space change descriptor*, with $el_{150\%}$ being calculated in the same way as in Chapter 3.3.2, as follows:

$$SoCME_{\mathcal{MS}} = \frac{el_{\mathcal{F}}}{el_{150\%}}.$$

The AV and PV contributions to the variant space change descriptor are calculated as

$$SoCME^{AV}_{\mathcal{MS}} = \frac{el^{AV}_{\mathcal{F}}}{el_{150\%}}$$

and

$$SoCME_{\mathcal{MS}}^{PV} = \frac{el_{\mathcal{F}}^{PV}}{el_{150\%}}.$$

Since every changed delta operation is classified either as AV or as PV, the following holds:

$$SoCME_{\mathcal{MS}} = SoCME_{\mathcal{MS}}^{AV} + SoCME_{\mathcal{MS}}^{PV}.$$

In addition, the relative distribution of the changed operations is calculated independently of the size of the 150% model:

$$AV_{\mathcal{MS}} = \frac{el_{\mathcal{F}}^{AV}}{el_{\mathcal{F}}}, \qquad PV_{\mathcal{MS}} = \frac{el_{\mathcal{F}}^{PV}}{el_{\mathcal{F}}}.$$

Algorithm 1 DFS-Based Splitting of a Delta Derivation Tree

Require: Delta derivation tree $T = (V, E, \lambda, \sigma, \mu, v_0)$
Ensure: Split-induced subtree forest $\mathcal{F}$ (Set of subtrees)

$\mathcal{F} \leftarrow \emptyset$
create a new subtree T_1 with root v_0
add T_1 to $\mathcal{F}$
procedure DFS-Split($w, v, \delta, T_{\text{cur}}$)
    if $v \neq \text{root}(T_{\text{cur}})$, $\mu(v)$ is defined **and** HasChanged(w, v, δ) **then**
        create a new subtree T_{new} with root v
        add T_{new} to $\mathcal{F}$
        $T_{\text{cur}} \leftarrow T_{\text{new}}$
        $chg_{\mu(v)} \leftarrow$ ReduceChange(w, v, δ)
    end if
    for all $(v, u) \in E$ **do**
        $\delta' \leftarrow \delta \cdot \lambda(v, u)$
        add node u and edge (v, u) to T_{cur}
        if $\mu(v)$ is defined **then**
            DFS-Split($v, u, \delta', T_{\text{cur}}$)
        else
            DFS-Split($w, u, \delta', T_{\text{cur}}$)
        end if
    end for
end procedure

DFS-Split($v_0, v_0, \emptyset, T_1$)
return $\mathcal{F}$

Algorithm 2 HasChanged

Require: Two nodes (w, v), delta δ, delta model $DM^{t=1}$
Ensure: true or false

```
1: if chg_δ ≠ ∅ then
2:     return true
3: end if
4: return false
```

Algorithm 3 Reduce the Change

Require: Two nodes (w, v), delta δ delta model $DM^{t=1}$

```
1: procedure ReduceChange(w, v, δ)
2:     for all op deleted during reduce(σ^{t=1}_{μ(w)} · δ) do
3:         if op ∈ chg_δ then
4:             delete op from chg_δ
5:         end if
6:     end for
7:     return chg_δ
8: end procedure
9:
10: return ReduceChange(w, v, δ)
```

Algorithm 4 Counting Valid Model Variants in a Delta Derivation Tree

Require: Delta derivation tree $T = (V, E, \lambda, \sigma, \mu, v_0)$
Ensure: Number of valid model variants in T

```
1: procedure CountVariants(v)
2:     c ← 0
3:     if μ(v) is defined then
4:         c ← c + 1
5:     end if
6:     for all (v, u) ∈ E do
7:         c ← c + CountVariants(u)
8:     end for
9:     return c
10: end procedure
11:
12: return CountVariants(v_0)
```

Algorithm 5 Counting Changed Delta Operations in $\mathcal{MS}^{t=1}$

Require: Set of delta derivation trees $\mathcal{F} = \{T_1, \ldots, T_k\}$, delta model DM and $DM^{t=1}$

Ensure: Number of changed delta operations and their AV/PV decomposition for all valid model variants in $\mathcal{F}$

1: $el_{\mathcal{F}} \leftarrow 0$
2: $el_{\mathcal{F}}^{AV} \leftarrow 0$
3: $el_{\mathcal{F}}^{PV} \leftarrow 0$
4: **for all** $T \in \mathcal{F}$ **do**
5: $\quad n \leftarrow \textsc{CountVariants}(T)$
6: $\quad el_{\mathcal{F}} \leftarrow el_{\mathcal{F}} + (|chg_{\mu(v_0 \in T)}| \times n)$
7: $\quad el_{\mathcal{F}}^{AV} \leftarrow el_{\mathcal{F}}^{AV} + (|chg_{\mu(v_0 \in T)}^{AV}| \times n)$
8: $\quad el_{\mathcal{F}}^{PV} \leftarrow el_{\mathcal{F}}^{PV} + (|chg_{\mu(v_0 \in T)}^{PV}| \times n)$
9: **end for**
10: **return** $(el_{\mathcal{F}}, el_{\mathcal{F}}^{AV}, el_{\mathcal{F}}^{PV})$

Running Example

We calculate the *variant space change descriptor* for our running example, by firstly running algorithm 1 on our derivation Tree $T^{t=1}$. We get the following set of subtrees, which are illustrated in Figure 3.2:

$$\mathcal{F} = \{T_1, T_2, T_3, T_4\}.$$

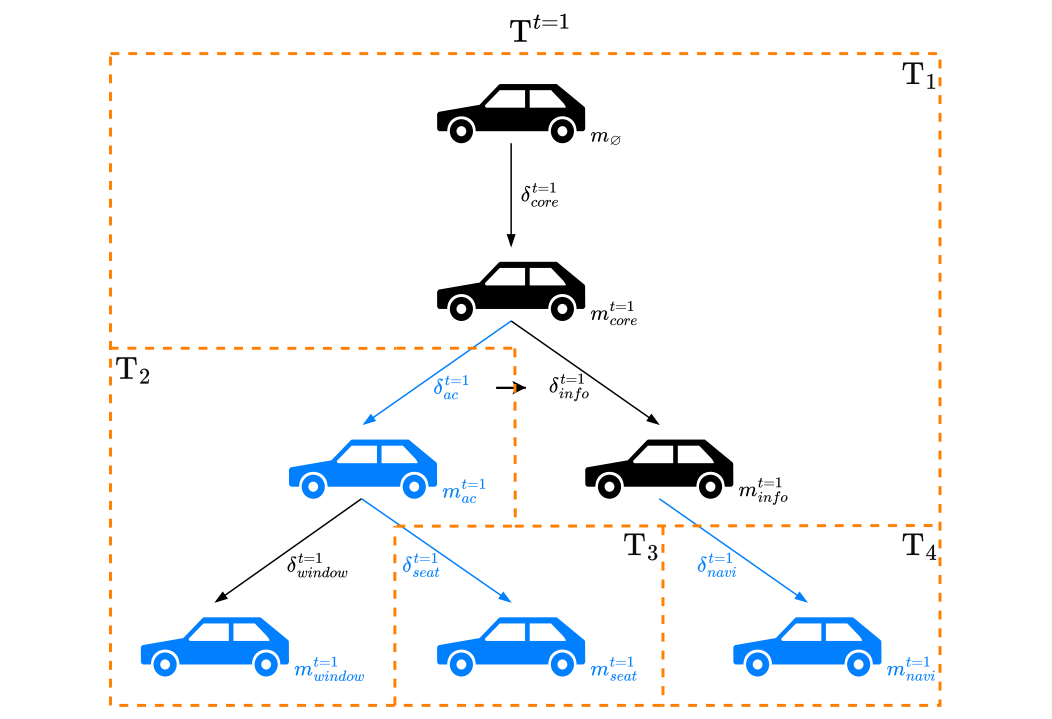


Figure 3.2: Running Example Algorithm 1

Except for the initial subtree T_1, we can see that each subtree begins with a model variant that follows directly after a changed delta. We can now run Algorithm 5 on $\mathcal{F}$, which uses Algorithm 4 to count the valid model variants in each subtree, multiply them by the changed modelling elements of each root model variant and then aggregats them. Thus, we obtain the *variant space change descriptor* as follows:

$$\textsc{CountVariants}(T_1) = 3, \textsc{CountVariants}(T_2) = 2,$$
$$\textsc{CountVariants}(T_3) = 1, \textsc{CountVariants}(T_4) = 1,$$

$$SoCME_{\mathcal{MS}} = \frac{\sum_{T_i \in \mathcal{F}} \textsc{CountVariants}(T_i)|chg_{\mu(v_0 \in T_i)}|}{el_{150\%}}$$
$$= \frac{(3 \times 0) + (2 \times 2) + (1 \times 2) + (1 \times 20)}{216} = \frac{26}{216}.$$

We obtain the same result for $SoCME_{\mathcal{MS}}$ as for $SoCME^{reduced}_{\mathcal{MS}}$, shown in the running example in Chapter 3.3.2. Therefore, the result is the same whether we use the orange path or the green path with reduction.

3.4 Summary

First, Delta-Modelling and higher-order Delta-Modelling were formalised together with the corresponding application functions, delta sequences, and model variant spaces. Based on these preliminaries, we then formalised how the evolution of an individual model variant can be described by diffing delta sequences, projecting higher-order deltas based on a specific model variant, reducing redundant delta operations, and calculating the *variant change descriptor*. This enables us the to estimate the effort and risk introduced by the evolution for each model variant individually.

Building on this, we addressed the evolution of the entire model variant space. For this purpose, the *variant space change descriptor* was defined. In addition, the 150% model, delta derivation trees and split-induced subtree forests were introduced as the structural basis for the proposed algorithms, illustrating two different paths to calculate the *variant space change descriptor*. One that uses the calculations for the *variant change descriptor*, and one that calculates the *variant space change descriptor* directly. Overall, we provide the formal concepts and algorithms required to describe both individual model variant evolution and the evolution of the complete model variant space with the description model of SGE, thereby enabling the estimation of effort and risk caused by higher-order delta evolution.

4. Implementation

This chapter describes the implementation of the concepts introduced in Chapter 3. While the design chapter defines the descriptors and algorithms independently of a specific modelling technology, the implementation must operate on the existing EMF-based delta repositories in the Body Comfort System (BCS) case study. Consequently, the implementation is not introduced for the calculations. Instead, the generated generic delta metamodel and the generated state-machine delta metamodel are reused directly, since state-machine delta models are used for the evaluation process. The implemented classes are located in the module `ChangeDescriptionOfEvolvedProductLine`.

Chapter 4.1 introduces the existing implementation required for the implementation of this thesis, as well as delta repositories, which are implementation artefacts that realise model variants. We then explain the differences between each implementation and the corresponding concept in the design chapter: Delta Diffing with `DeltaRepositoryDiff`, Reduction with `DeltaRepositoryReducer`, Variant Change Descriptor with `VariantChangeDescriptor`, Aggregated Variant Change Descriptor with `AggregatedVariantChangeDescriptor` and Variant Space Change Descriptor with `VariantSpaceChangeDescriptor`.

4.1 Existing Implementations

4.1.1 Body Comfort System Repository

This implementation uses Neumann's implementation[1] of a generic and state-machine specific delta dialect for the delta model is used for this implementation. The delta repository design is also used and explained below. It contains deltas and delta operations that conform to the generic and state-machine-specific delta dialect.

A delta repository (more specifically, a `DeltaRepository` instance) is an implementation artefact that represents a specific model variant. Each repository contains a list of `Delta` objects, and each delta contains a list of `DeltaOp` objects. In order to

[1] https://gitlab.kit.edu/kit/tva/neumann/convide-b04/bcs/-/tree/BCS_ecore?ref_type=heads

represent a specific model variant in our repository, we must design our delta model with an empty core. We then add the initial core as a delta. When we derive the delta sequence of a model variant m_i containing the core delta in it, we can apply the delta sequence to an empty model to produce the model variant m_i. The deltas of the model variant m_i are now saved in the delta repository $repo_{m_i}$ alongside their after clauses and application conditions, enabling us to derive a valid delta sequence for the model variant m_i. We now define the universe of all delta repositories , denoted by $REPO_{\mathcal{M}}$, as as the realisation of all valid models that can be created based on $\mathcal{E}$ and $\mathcal{R}$, thus $\mathcal{M}$. For each model variant $m_i \in \mathcal{M}$ there is a delta repository $repo_{m_i} \in REPO_{\mathcal{M}}$. The empty model $m_\varnothing \in \mathcal{M}$ can be represented as an empty repository $repo_{m_\varnothing}$. Based on this definition, we can substitute a model variant with its respective delta repository in any function that uses a model variant.

4.1.2 Master's Thesis of Simon Bothe

This repository[2] provides the implementation of product and delta case studies for applying the Delta-CPR testing procedure. It is part of Simon Bothe's master's thesis at TVA (KIT). It includes the delta dialects introduced in the previous chapter, and the repository was used as the basis for the implementation. Additionally, it contains tools for creating delta repositories from the BCS Study, which are used in the evaluation chapter.

4.2 Delta Diffing with `DeltaRepositoryDiff`

The class `DeltaRepositoryDiff` implements the delta diffing function described in Chapter 3.2.1. The design describes the diff is described as an alignment-based comparison of delta sequences and delta operations. However, in the implementation, the input is not represented as two arbitrary mathematical sequences, but as two `DeltaRepository` instances. Each repository contains a list of `Delta` objects, and each delta contains a list of `DeltaOp` objects. Therefore, the implemented diff therefore performs the alignment in two stages: first on deltas and then on operations inside shared deltas.

At the delta level, deltas are matched by name. The method that prepares this step builds a map from each delta name to the corresponding `Delta`. Since a delta repository may contain multiple deltas with the same name, the implementation internally adds an occurrence suffix internally when duplicate names are encountered. This preserves deterministic matching while avoiding accidental overwriting in the map. If a delta exists only in the base repository, all of its operations are translated into corresponding removal operations in the diff. If a delta only exists in the evolved repository, all of its operations are copied into the diff as additions. If a delta exists in both repositories, the operations of the two deltas are compared.

At the operation level, the implementation replaces the abstract operation alignment B with signature-based matching. Exact matches are removed first. For this purpose, an operation signature is constructed from the classifier name and a selected set of structural features, such as `idOfAffectedObject`, `modifiedAttString`, `modifiedRefString`, `idOfAddedValue`, `idOfRemovedValue`, `idOfNewValue`, `newValue`,

[2] https://gitlab.kit.edu/kit/tva/neumann/studenttheses/ma_simon_bothe

index, and value. This is an implementation-specific adaptation of the mathematical equality check.

Once the exact matches have been removed, the implementation handles a restricted form of substitutions. In the design, substitutions are defined as operation-level differences between aligned operations. In the concrete state-machine delta metamodel, a modification is represented by generated classes such as Set... and Unset.... These operations may change their value while still referring to the same affected object and the same modified attribute or reference. The implementation therefore defines a replacement key for operations whose classifier name starts with Set or Unset. This replacement key ignores the concrete value, keeping only the replacement identity, which consists mainly of the classifier name, affected object, and modified feature. If an unmatched operation from the base repository and an unmatched operation from the evolved repository have the same replacement key, the diff contains a removal of the old operation and an addition of the new operation.

The resulting diff is written as a new DeltaRepository with one synthetic Delta named Diff Delta. This differs from the design notation, where the diff is described as a delta sequence. The single-delta representation is sufficient for the later descriptor calculations, because these steps only require the set and number of changed operations.

Creating a removal operation is also implementation-specific. The generated state-machine delta metamodel contains separate classes for addition and removal operations. Therefore, DeltaRepositoryDiff derives possible removal classifier names from the classifier name of the original operation. For example, operations starting with CreateAndAddByValueInto... are mapped to matching DestroyByValueFrom... operations, while those starting with AddByValueInto are mapped to matching RemoveByValueFrom... operations. For Set... operations, the implementation creates a corresponding Unset... operation where possible. Relevant features, such as affected object IDs and modified feature names, are copied into the generated removal operation.

4.3 Reduction with DeltaRepositoryReducer

The reduction described in Chapter 3.2.4 is implemented by the DeltaRepositoryReducer class. Its purpose is to remove redundant operations before descriptor values are calculated. The implementation provides two variants: repository-wide reduction and single-delta reduction. Repository-wide reduction is used when reducing a complete model variant repository. The single-delta reduction is used when a delta is reduced in isolation.

First, the reducer creates a copy of the input repository or delta. This is important because the original input resources are also used by other calculations and must not be modified as a side effect. The actual reduction is then performed in place on the copy. The implementation repeatedly applies the reduction rules until a fixed point is reached. This fixed-point iteration is necessary because applying one rule may make another rule applicable. For example, removing an add-remove pair could bring two modification operations closer together, making an overwritten modification removable.

The first implemented rule is add-remove elimination. Addition-like operations are recognised by classifier names starting with `CreateAndAddByValueInto`, `CreateAndInsertByIndexInto`, or `AddByValueInto`. Removal-like operations are recognised by classifier names starting with `DestroyByValueFrom`, `DestroyByIndexFrom`, `DestroyAllFrom`, or `RemoveByValueFrom`. The created or removed element is identified by the `idOfAddedValue` or `idOfRemovedValue`. If an addition and a later removal refer to the same element, both operations are deleted. Additionally, any operations between them that target the same element are also removed, as these only affect an element that does not exist in the final reduced result.

The second implemented rule is overwritten modification elimination. Modification operations are recognized by classifier names starting with `Set`, `Unset`, `AddByValue`, `RemoveByValue`, or `Clear`. The implementation determines a target key from features such as `idOfAffectedObject`, `idOfAddedValue`, `idOfRemovedValue`, and `idOfNewValue`. The modified scope is determined primarily through `modifiedRefString`; if that feature is not present, the scope is derived from the classifier name or from the operation kind. If two later operations affect the same target and the same scope, the earlier operation is deleted, provided that no operation between them affects the same target and scope.

The implementation is intentionally conservative. A rule is only applied if the classifier name and the relevant structural features provide a clear match. If an operation does not expose the expected feature, or if the target cannot be determined, it is kept. This differs from the mathematical design, where operations are assumed to have a well-defined target and scope. The conservative strategy is necessary because the concrete delta metamodel contains many generated operation classes with different feature sets, and removing an operation without a reliable match could change the semantics of the resulting variant.

For repository-wide reduction, all operations are flattened into one operation sequence while retaining a reference to the owning delta. This corresponds to the design's view of a delta sequence as the operations applied along a derivation path. When operations are deleted, the reducer removes them from their original owning deltas. This preserves the repository structure and ordering of the remaining operations. The implementation therefore reduces the effective operation sequence without reconstructing the whole delta model from scratch.

4.4 Variant Change Descriptor

The class `VariantChangeDescriptor` implements the descriptor for an individual model variant. In the design, the variant-specific change chg_{m_i} is obtained by diffing the delta sequence before and after the higher-order delta evolution. In the implementation, the `VariantChangeDescriptor` takes two repositories as input: the base model variant repository and the evolved model variant repository.

The computation consists of three steps. First, the implementation uses the `DeltaRepositoryDiff` class to compute the diff repository between the base and evolved model variant repositories. Second, it counts the relevant operations in the diff repository. This value corresponds to $|\delta_{chg_{m_i}}|$ in the design. Third, the relevant operations in

the evolved model variant repository are counted. This value corresponds to el_{m_i}. The share of changed modelling elements is then calculated as

$$SoCME_{m_i} = \begin{cases} 0 & \text{if } el_{m_i} = 0, \\ \frac{|\delta_{chg_{m_i}}|}{el_{m_i}} & \text{otherwise.} \end{cases}$$

In addition to the share of changed modelling elements, the implementation classifies the changed delta operations. This is realised by an explicit mapping from delta operation class names to one of two variation classes: `AV` for AV and `PV` for PV.

After the diff repository has been computed, the implementation iterates over the relevant changed operations and looks up each operation's class name in the AV/PV mapping. Technical initialization operations and destroy or remove operations are excluded from the `SoCME` count, as in the original calculation. For every counted operation, the implementation increments either the AV counter or the PV counter. The relative shares are then calculated as

$$AV_{m_i} = \begin{cases} 0 & \text{if } |\delta_{chg_{m_i}}| = 0, \\ \frac{|\delta^{AV}_{chg_{m_i}}|}{|\delta_{chg_{m_i}}|} & \text{otherwise,} \end{cases} \qquad PV_{m_i} = \begin{cases} 0 & \text{if } |\delta_{chg_{m_i}}| = 0, \\ \frac{|\delta^{PV}_{chg_{m_i}}|}{|\delta_{chg_{m_i}}|} & \text{otherwise.} \end{cases}$$

If a relevant changed operation has no entry in the mapping, the implementation throws an exception. This ensures that newly introduced delta operation types are not silently ignored or assigned to an arbitrary variation class. The command-line output therefore contains the share of changed modelling elements together with the AV/PV distribution, for example `60% AV` and `40% PV`.

The implementation also writes the AV/PV mapping of the computed diff to a CSV file. Each row contains the delta name, the operation class, the operation identifier, the operation name, the assigned impact classification, and a flag indicating whether the operation was counted in the `SoCME` calculation. Destroy and remove operations are included in this file for traceability, but are marked as not counted.

The implementation returns the generated diff repository, the number of changed operations and evolved model variant operations, the calculated share, and the AV/PV counts and shares.

The largest practical change compared to the design is that the implementation does not explicitly construct a projected higher-order delta. The design uses the projected higher-order delta to identify which part of the higher-order evolution affects one model variant. In the implementation, this projection is realised implicitly by first constructing variant-local repositories and then diffing only these repositories. Thus, the repository extraction step takes over the role of the projection.

The implementation also separates diffing from reduction. The `VariantChangeDescriptor` itself computes the raw share for the two repositories it receives. If reduced descriptor values are required, the caller reduces the base and evolved model variant repositories before invoking the descriptor.

4.5 Aggregated Variant Change Descriptor

The class `AggregatedVariantChangeDescriptor` implements the aggregation approach described in Chapter 3.3.2. It receives a list of model variant pairs. Each pair consists of one base model variant repository and one evolved model variant repository. For each pair, the implementation calculates the number of changed operations and modelling elements in the evolved model variant. These values are then summed over all model variants.

The aggregation follows the equation from the design chapter:

$$SoCME_{\mathcal{MS}} = \frac{\sum_{m_i \in \mathcal{MS}} |\delta_{chg_{m_i}}|}{el_{150\%}}.$$

In addition to the overall share of changed modelling elements, the implementation also aggregates the AV/PV variation classification of the changed operations. For every model variant pair, the implementation computes the variant-local diff repository using `DeltaRepositoryDiff`. The changed operations in this diff are then classified with the same AV/PV mapping used by `VariantChangeDescriptor`. The mapping assigns each relevant delta operation either to an AV or to a PV.

For each variant pair, the implementation stores the number of changed AV operations and changed PV operations in the corresponding `VariantResult`. These counts are also accumulated globally as `totalAttributeVariationOperations` and `totalPrincipleVariationOperations`. The implementation then calculates the AV and PV contributions to the aggregated descriptor as

$$SoCME^{AV}_{\mathcal{MS}} = \frac{\sum_{m_i \in \mathcal{MS}} |\delta^{AV}_{chg_{m_i}}|}{el_{150\%}}$$

and

$$SoCME^{PV}_{\mathcal{MS}} = \frac{\sum_{m_i \in \mathcal{MS}} |\delta^{PV}_{chg_{m_i}}|}{el_{150\%}}.$$

The relative AV/PV distribution of the aggregated change is calculated as

$$AV_{\mathcal{MS}} = \frac{\sum_{m_i \in \mathcal{MS}} |\delta^{AV}_{chg_{m_i}}|}{\sum_{m_i \in \mathcal{MS}} |\delta_{chg_{m_i}}|}$$

and

$$PV_{\mathcal{MS}} = \frac{\sum_{m_i \in \mathcal{MS}} |\delta^{PV}_{chg_{m_i}}|}{\sum_{m_i \in \mathcal{MS}} |\delta_{chg_{m_i}}|}.$$

The implementation supports two calculation modes. In `RAW` mode, the model variant repositories are used as loaded. In `REDUCED_VARIANTS` mode, both the base and evolved repositories of every model variant pair are reduced with `DeltaRepositoryReducer` before the diff is computed. This reduced approach aligns with the design discussion that unreduced deltas can only provide an upper bound for the share of changed modelling elements. By reducing the variant-local repositories before diffing and counting, redundant operations are removed.

Another implementation detail is the creation of model variant repositories. The aggregated variant change descriptor requires explicit repositories for each model variant. These repositories can be produced by using the `VariantEvaluationRepositoryExporter`. The exporter reads a file that defines valid model variant pairs in the following format:

```
M: base=[...] -> evolved=[...].
```

For each model variant listed, the exporter builds feature-selection graphs for the base and evolved model variant repositories. It then finds the node with the requested feature selection, extracts the active deltas for that node, and saves the resulting repositories. These exported repositories are saved unreduced by design. This makes it possible to compare raw and reduced aggregation modes using the same input data.

Therefore, the aggregation implementation realises the green path from the design chapter: it explicitly enumerates the model variants, computes a variant change descriptor for each of them, and aggregates the results using the 150% model.

4.6 Variant Space Change Descriptor

The class `VariantSpaceChangeDescriptor` implements the descriptor for the complete model variant space. It is the orange path in Chapter 3.3.3, because it derives model variants from the delta repository and its feature selections instead of requiring pre-exported model variant repositories. However, the implementation uses a pragmatic feature-selection graph rather than a direct implementation of the split-induced subtree forest algorithm.

The graph construction is implemented in `FeatureSelectionDerivationGraphVisualizer`. Each node in the graph represents a feature selection. Each node stores the selected features, the closure of selected and derived features, and the active deltas for this state. Edges represent the selection of one additional selectable feature. The graph is constructed through a breadth-first exploration of selectable features. A state is only accepted if all active deltas satisfy their `after` dependencies and if the active deltas can be topologically ordered. Thus, the ordering constraints from the design are implemented through dependency validation and topological sorting.

The feature model used by the graph is inferred from the applicability conditions of the deltas. Literal applicability conditions identify selectable or derived features. Top-level `OrExpression` conditions are interpreted as alternatives, and the implementation adds pairwise exclusions between their literal operands. Derived features are inferred from dependency patterns: if a feature is always introduced through deltas that depend on other feature-specific deltas, the implementation treats it as derived and closes it automatically when its providers are selected.

For each valid graph node, the `VariantSpaceChangeDescriptor` extracts the active deltas from the base and evolved model variant repositories. The active deltas are selected by name and copied into a new temporary `DeltaRepository`. Both active repositories are then reduced. Afterwards, the variant-local diff is computed using `DeltaRepositoryDiff` and the relevant operations in the diff and evolved active

repository are counted. The implementation accumulates these counts in `elF` and `el150`. The variant space change descriptor is then calculated as

$$SoCME_{\mathcal{MS}} = \begin{cases} 0 & \text{if } el_{150\%} = 0, \\ \frac{el_{\mathcal{F}}}{el_{150\%}} & \text{otherwise.} \end{cases}$$

In addition to counting the changed operations, the implementation classifies the changed operations in each variant-local diff into AVs and PVs. For this, `VariantSpaceChangeDescriptor` reuses the AV/PV impact mapping implemented in `VariantChangeDescriptor`. For every computed diff repository, the implementation calls the shared impact-distribution calculation and obtains the number of changed `AV` operations and changed `PV` operations. These values are accumulated in `elFAV` and `elFPV`. The AV and PV contributions to the variant space change descriptor are then calculated as

$$SoCME^{AV}_{\mathcal{MS}} = \begin{cases} 0 & \text{if } el_{150\%} = 0, \\ \frac{el^{AV}_{\mathcal{F}}}{el_{150\%}} & \text{otherwise,} \end{cases}$$

and

$$SoCME^{PV}_{\mathcal{MS}} = \begin{cases} 0 & \text{if } el_{150\%} = 0, \\ \frac{el^{PV}_{\mathcal{F}}}{el_{150\%}} & \text{otherwise.} \end{cases}$$

Since every relevant changed operation is classified either as `AV` or as `PV`, these two values decompose the overall descriptor:

$$SoCME_{\mathcal{MS}} = SoCME^{AV}_{\mathcal{MS}} + SoCME^{PV}_{\mathcal{MS}}.$$

The implementation also calculates the relative distribution of the changed operations:

$$AV_{\mathcal{MS}} = \begin{cases} 0 & \text{if } el_{\mathcal{F}} = 0, \\ \frac{el^{AV}_{\mathcal{F}}}{el_{\mathcal{F}}} & \text{otherwise,} \end{cases} \qquad PV_{\mathcal{MS}} = \begin{cases} 0 & \text{if } el_{\mathcal{F}} = 0, \\ \frac{el^{PV}_{\mathcal{F}}}{el_{\mathcal{F}}} & \text{otherwise.} \end{cases}$$

The class provides two calculation modes. The first mode derives model variants automatically from the evolved repository's feature-selection graph and evaluates the same feature selection in the base and evolved repositories. The second mode reads a model variant definition file. This mode is necessary for evolutions where a model variant changes its feature selection during evolution, for example from `base=[Seat]` to `evolved=[Seat, AirConditioning]`. In this case, the base graph and evolved graph are built separately, and each side of the model variant pair is resolved against the corresponding graph.

This implementation differs from the design algorithm in one important respect. Rather than avoiding visiting every individual model variant, the design splits the derivation tree into subtrees, multiplying the changed operations at each subtree root by the number of valid variants below it. However, the implemented `VariantSpaceChangeDescriptor` still iterates over the valid graph states and computes a variant-local diff for each state. This is equivalent to the aggregation formula

for the evaluated repositories, but it is less optimised than the subtree-counting algorithm. This implementation was chosen because it is easier to validate against the explicit aggregation implemented by `AggregatedVariantChangeDescriptor`. It also provides detailed per-variant contributions, including AV/PV distributions, which are useful for verifying the evaluation results.

4.7 Summary

The implementation realises the design concepts on top of the existing delta infrastructure. `DeltaRepositoryDiff` implements the operation-level diff by matching deltas by name and operations by stable signatures rather than by object identity. `DeltaRepositoryReducer` implements the reduction rules conservatively by inspecting the names and features of the generated operation classes. `VariantChangeDescriptor` calculates the share of changed modelling elements for one model variant by diffing variant-local repositories. `AggregatedVariantChangeDescriptor` explicitly aggregates these variant change descriptors and supports both raw and reduced calculations. Finally, `VariantSpaceChangeDescriptor` derives valid variants from feature-selection graphs, extracts active deltas, reduces them, and calculates the descriptor for the complete model variant space.

The main implementation adaptations are caused by the concrete representation. Deltas and operations must be loaded from serialised resources, operation equality has to be approximated by structural signatures, and valid model variants have to be reconstructed from applicability conditions and `after` dependencies. These adaptations preserve the the design's intent while making the descriptors executable for the BCS case study repositories.

5. Evaluation

In this chapter, we evaluate our sub-goals. SG1 involves calculating the *variant change descriptor*. SG2a involves aggregating the *variant change descriptor* in order to calculate the *variant space change descriptor*, while SG2b involves calculating the *variant space change descriptor* directly using the underlying structure of the delta model. We evaluate the following aspects:

(1) For SG1, we evaluate the correctness of the derived variant-specific changes from higher-order deltas. We demonstrate this correctness, because the calculation of variant-specific changes forms the basis of the calculation of the *variant change descriptor* and *variant space change descriptor*. Therefore, if the variant-specific changes are incorrect, the change descriptors will also be incorrect.

(2) For SG1, we evaluate the technical applicability of the *variant change descriptor*. We demonstrate that the change descriptors, introduced by Ochs et al. [15], can be applied in our context to generate the *variant change descriptors*.

(3) For SG2a, we evaluate the technical applicability of the *variant space change descriptors*. We shows whether the *variant change descriptor* obtained for individual model variants can be combined into one descriptor that characterises the evolution of the evaluated model variant space.

(4) For SG2b, we evaluate the feasibility of directly calculating the *variant space change descriptor*, rather than calculating the *variant change descriptor* for each model variant. This direct calculation is compared with the aggregation-based calculation from SG2a. The relevant question is whether both calculation paths describe the same evaluated model variant space and therefore yield the same descriptor values.

Our evaluation is based on a tailored version of the BCS case study introduced in Chapter 5.1. Chapter 5.2, 5.3, 5.4 and 5.5 evaluate (1), (2), (3) and (4) respectively. Chapter 5.6 discusses the results and limitations of the evaluation. Finally, Chapter 5.7 summarises the evaluation.

5.1 Subject System

This section introduces the BCS case study and explains how it is tailored into a subject system that is suitable for evaluating the proposed descriptors. The subject system must satisfy two requirements. First, it must contain enough variability and evolution to exercise the calculations introduced in this thesis. Second, it must remain small enough for the expected results to be derived manually as ground truth.

5.1.1 Body Comfort System Case Study

The BCS is an academically used case study representing a simplified, yet realistic, embedded automotive SPL with rich variability [11]. It consists of distributed electronic control units (ECUs), each of which manages a subsystem of the BCS, such as a door control, a Human-Machine Interface (HMI), an alarm system, and a central locking system. Each ECU communicates with others via a Controller Area Network (CAN) bus.

The BCS is suitable for this evaluation because an SPL-based version of the case study is available and because it has already been used in the context of delta-oriented modelling. In its existing form, the case study provides a feature model with 27 features, 11.616 valid product variants (the case study focuses on a representative subset of 17 product variants) and five application constraints, and a state-machine based delta model consisting of 15 deltas. It therefore provides a realistic basis for generating different model variants. In addition, four existing evolution scenarios extend the BCS with further functionality, such as a wiper system, electric seat adjustment, heatable windows and automatic headlights [13]. These evolution scenarios are relevant for this thesis because they provide higher-order delta changes that can be applied to existing BCS delta models.

5.1.2 Tailoring the BCS Case Study as Subject System

Figure 5.1 and 5.2 illustrate the base and evolved delta models of our subject system, which are extracted from the existing state machine based delta model of the BCS. The features and corresponding deltas used from the BCS are shown in Table 5.1. The subject system is intentionally smaller than the full BCS. This reduces the manual effort needed to derive the ground truth, while still covering the cases required by the evaluation. The features were selected for the following reasons:

Reuse of Existing implementation: Every delta, except for `AirConditioning`, had already been implemented in a delta repository, including the application conditions and after clauses[1]. Thus, these could easily be used to generate different model variants.

Coverage of Evaluation Cases: The subject system must contain redundant delta operations because the evaluation investigates the difference between raw and reduced calculations. The newly implemented `AirConditioning` feature introduces additional state-machine behaviour and is used to create cases in which redundant operations affect the descriptor calculation. This allows us to evaluate whether reducing the variant-specific changes before calculating the descriptors changes the measured share of changed modelling elements.

[1] https://gitlab.kit.edu/kit/tva/neumann/studenttheses/ma_simon_bothe

Valid Configurations Through Feature Dependencies: The evaluated feature selections must satisfy the dependency constraints of the BCS feature model. Therefore, some features are selected explicitly, while others are derived automatically as dependent features. For example, selecting the `Wiper` feature requires a sensor feature, such as `LowQualitySensor` or `HighQualitySensor`. Similarly, selecting `StatusLED` requires the dependent features `LEDClean` and `LEDWiper`. In the following tables, *selected* features denote explicitly chosen features, while *closed* features denote the complete feature selection after adding the dependent features.

Feature	Deltas
`Wiper`	DAddWiperSignals, Delta_Wiper_Core
`LowQualitySensor`	Delta_Medium_LowSensor
`HighQualitySensor`	DAddHighQualitySensorSignals, Delta_HighSensor*
`Seat`	DAddSeatSignals, DAddSeat, DAddSeatSet, DAddSeatPos
`AutomaticHeadlights`	DAddLightSignals, DAddLight*
`Clean`	Delta_Clean, Delta_Clean_Counter
`ManualPowerWindow`	DAddPWSignals, DAddManPWSignals, DAddManPWCore
`StatusLED`	DAddStatusLED
`LEDClean`	DAddStatusLEDClean
`LEDWiper`	DAddStatusLEDWiper
`AirConditioning`	DAddAirConditioning (Implementation in Appendix A.1)

Table 5.1: Overview of used Features and Deltas

Table 5.1 lists the features used in the subject system and the deltas that implement them in the delta repository. The table therefore defines which delta layers can become active for the evaluated variants. Features such as `Wiper`, `Seat` and `AutomaticHeadlights` are used to construct the base derivation path, while `AirConditioning`, `HighQualitySensor` and `StatusLED` are used in the evolved derivation path.

Table 5.2 summarises the subject system in numbers. The table distinguishes between explicitly selected features, dependent features derived through feature closure, exported model variant repositories and the complete base and evolved delta repositories. The evaluated model variant pairs are the five pairs obtained by comparing MV_i with $MV_i^{t=1}$ for $i \in 1, \ldots, 5$.

Figure 5.1 shows the base derivation path. The label inside the box identifies the variant, while the text below the label summarises the main feature layer represented by that variant. The base delta model starts with the core variant MV_1 and then derives larger variants by successively adding feature layers: wiper behaviour, seat behaviour, automatic headlights, and finally clean plus manual-power-window

Measure	Value	Explanation
Features explicitly selected in the evaluation pairs	7	`Wiper`, `AirConditioning`, `Seat`, `AutomaticHeadlights`, `Clean`, `ManualPowerWindow` and `StatusLED`.
Implicitly derived features	4	`HighQualitySensor`, `LowQualitySensor`,`LEDClean`, and `LEDWiper`.
Delta repositories used for the variant-pair evaluation	10	Five base and five evolved variant repositories exported from the complete repositories.
Evaluated model variant pairs	5	The pairs shown in Figure 5.1 and Figure 5.2.
Deltas in the complete base / evolved repository	19 / 20	The evolved repository adds the delta `DAddAirConditioning`.
Delta operations in the complete base / evolved repository	1058 / 1306	The evolved repository has more features.
Delta operations in `DAddAirConditioning`	28	The newly introduced `DAddAirConditioning` delta.

Table 5.2: Subject system used in the evaluation

behaviour. Figure 5.2 shows the corresponding evolved derivation path after higher-order delta evolution. In the evolved model, `AirConditioning` is introduced in the first evaluated evolved variant. Consequently, every following evolved variant contains the air-conditioning layer in addition to the feature layers added along the derivation path. The evolved sequence then continues with the later feature additions and finally adds high-quality-sensor and status-LED behaviour for $MV_5^{t=1}$.

Each evaluated model variant pair compares one base variant from Figure 5.1 with the evolved variant at the same position in Figure 5.2. For example, MV_3 is compared with $MV_3^{t=1}$. This comparison captures the complete variant-specific evolution between both variants, not only the introduction of `AirConditioning`. Tables 5.3 and 5.4 describe the same variants in tabular form by listing their explicit feature selections, their closed feature selections and their active delta layers.

Variant	Explanation	Features	Delta layer
MV_1	core	selected: ∅; closed: ∅	`DAddCore`
MV_2	Adds wiper behavior	selected: Wiper; closed: Wiper, LowQualitySensor	`DAddWiperSignals; Delta_Wiper_Core; Delta_Medium_LowSensor`
MV_3	Adds seat behavior	selected: Wiper, Seat; closed: Wiper, Seat, LowQualitySensor	MV_2 `layer + DAddSeatSignals; DAddSeat; DAddSeatSet; DAddSeatPos`
MV_4	Adds automatic headlights	selected: Wiper, Seat, AutomaticHeadlights; closed: Wiper, Seat, AutomaticHeadlights, LowQualitySensor	MV_3 `layer + DAddLightSignals; DAddLight*`
MV_5	Adds clean manual-power-window behavior	selected: Wiper, Seat, AutomaticHeadlights, Clean, ManualPowerWindow; closed: Wiper, Seat, AutomaticHeadlights, Clean, ManualPowerWindow, LowQualitySensor	MV_4 `layer + DAddPWSignals; DAddManPWSignals; DAddManPWCore; Delta_Clean; Delta_Clean_Counter`

Table 5.3: Base Delta Model Variants

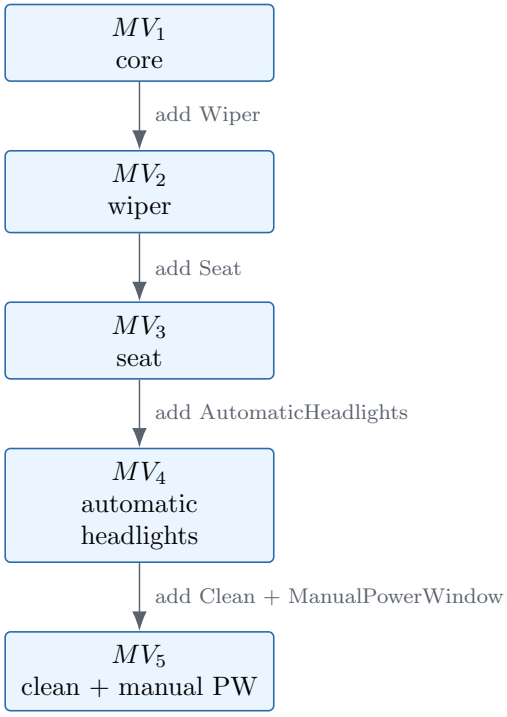


Figure 5.1: Base Delta Model

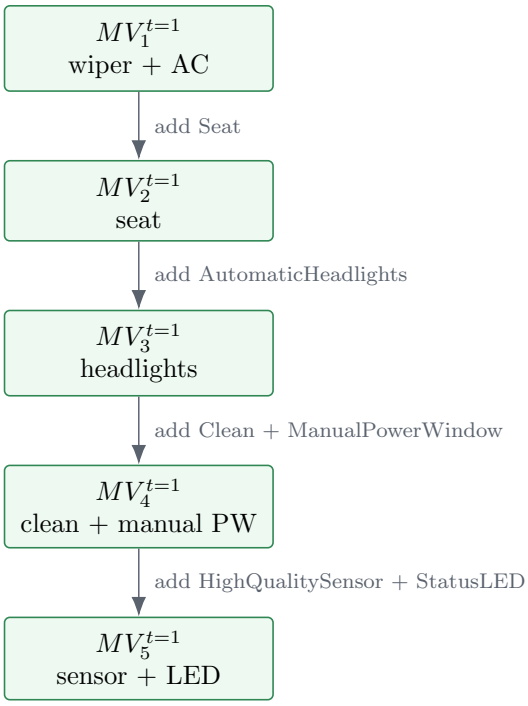


Figure 5.2: Evolved Delta Model

Variant	Explanation	Features	Delta layer
$MV_1^{t=1}$	Adds air conditioning to the wiper variant	selected: Wiper, AirConditioning; closed: Wiper, AirConditioning, LowQualitySensor	`DAddCore; DAddWiperSignals; Delta_Wiper_Core; Delta_Medium_LowSensor; DAddAirConditioning`
$MV_2^{t=1}$	Adds seat behavior	selected: Wiper, AirConditioning, Seat; closed: Wiper, AirConditioning, Seat, LowQualitySensor	$MV_1^{t=1}$ `layer + DAddSeatSignals; DAddSeat; DAddSeatSet; DAddSeatPos`
$MV_3^{t=1}$	Adds automatic headlights	selected: Wiper, AirConditioning, Seat, AutomaticHeadlights; closed: Wiper, AirConditioning, Seat, AutomaticHeadlights, LowQualitySensor	$MV_2^{t=1}$ `layer + DAddLightSignals; DAddLight*`
$MV_4^{t=1}$	Adds clean manual-power-window behavior	selected: Wiper, AirConditioning, Seat, AutomaticHeadlights, Clean, ManualPowerWindow; closed: Wiper, AirConditioning, Seat, AutomaticHeadlights, Clean, ManualPowerWindow, LowQualitySensor	$MV_3^{t=1}$ `layer + DAddPWSignals; DAddManPWSignals; DAddManPWCore; Delta_Clean; Delta_Clean_Counter`
$MV_5^{t=1}$	Adds high-quality sensor and status LED behavior	selected: Wiper, AirConditioning, Seat, AutomaticHeadlights, Clean, ManualPowerWindow, HighQualitySensor, StatusLED; closed: Wiper, AirConditioning, Seat, AutomaticHeadlights, Clean, ManualPowerWindow, HighQualitySensor, StatusLED, LowQualitySensor, LEDClean, LEDWiper	$MV_4^{t=1}$ `layer + DAddHighQualitySensorSignals; Delta_HighSensor*; DAddStatusLED; DAddStatusLEDClean; DAddStatusLEDWiper`

Table 5.4: Evolved Delta Model Variants

5.2 Variant-Specific Change from HO-Delta Evolution

This section evaluates whether the variant-specific changes caused by the higher-order delta evolution are derived correctly. The evaluation uses the five model variant pairs introduced in Chapter 5.1.2. Each pair compares one base model variant with its corresponding evolved model variant. More precisely, it compares the complete evolution of the selected base feature set to the selected evolved feature set. Each pair of base and evolved model variants from the subject system is compared with the ground truth to check for equality.

5.2.1 Setup

We use the complete base and evolved repositories (the repositories containing all the deltas of the base and evolved delta models) to create the model variant pairs. The markdown file describing the complete evaluation used in the implementation can looked up in the appendix A.1. The feature selections for each base and evolved model variant are given explicitly. The `VariantEvaluationRepositoryExporter` then exports one base and one evolved delta repository for each pair. During this export, the explicit feature selection is closed with the `FeatureSelectionDerivationGraph`. This is necessary because several features are not selected manually, but are implied by the feature dependencies of the BCS. For example, selecting `Wiper` derives `LowQualitySensor`, and selecting `StatusLED` derives `LEDClean` and `LEDWiper`.

The expected variant-specific changes are derived manually from the base and evolved delta repositories of the model variant pairs as the ground truth. We calculate the variant-specific changes for each pair of model variants by applying the `DeltaRepositoryDiff` class to each pair of model variants. The calculated variant-specific changes are then compared with the ground truth. This comparison checks whether the delta layers, additional behaviour and number of delta operations are equal.

5.2.2 Ground Truth

Table 5.5 shows the manually derived ground truth. The table lists the feature-level evolution, the corresponding delta layers that are expected to be newly relevant in the evolved model variant, and the number of changed delta operations in the resulting variant-specific change. Unchanged layers, such as the core delta or already existing wiper behaviour in later variants, are not repeated in the last column. The variant-specific changes themself can be seen in the appendix.

5.2.3 Results

The calculated variant-specific changes match the manually derived ground truth for all five variant pairs. The number of changed delta operations in each variant-specific change also matches the ground truth. The feature-closure step is essential for this result, because the expected low-quality sensor and LED-related deltas are not always explicitly selected by the input file. Table 5.6 summarises the comparison.

Variant pair	Base selection	Evolved selection	Changed ops	Expected variant-specific change
MV_1	$\emptyset$	`Wiper`, `AirConditioning`	232	Add wiper behaviour, the derived low-quality sensor behaviour, and air-conditioning behaviour.
MV_2	`Wiper`	`Wiper`, `AirConditioning`, `Seat`	187	Add air-conditioning behaviour and the seat delta layer.
MV_3	`Wiper`, `Seat`	`Wiper`, `AirConditioning`, `Seat`, `AutomaticHeadlights`	471	Add air-conditioning behaviour and the automatic-headlights delta layer.
MV_4	`Wiper`, `Seat`, `AutomaticHeadlights`	`Wiper`, `AirConditioning`, `Seat`, `AutomaticHeadlights`, `Clean`, `ManualPowerWindow`	266	Add air-conditioning behaviour, clean behaviour, and manual-power-window behaviour.
MV_5	`Wiper`, `Seat`, `AutomaticHeadlights`, `Clean`, `ManualPowerWindow`	`Wiper`, `AirConditioning`, `Seat`, `AutomaticHeadlights`, `Clean`, `ManualPowerWindow`, `HighQualitySensor`, `StatusLED`	250	Add air-conditioning behaviour, high-quality-sensor behaviour, and status-LED behaviour including the derived LED-clean and LED-wiper behaviour.

Table 5.5: Ground truth for variant-specific changes and changed delta operations

Variant pair	Ground-truth comparison	Changed ops	Result
MV_1	Wiper, low-quality sensor, and air-conditioning layers are present.	232	equal
MV_2	Air-conditioning and seat layers are present.	187	equal
MV_3	Air-conditioning and automatic-headlights layers are present.	471	equal
MV_4	Air-conditioning, clean, and manual-power-window layers are present.	266	equal
MV_5	Air-conditioning, high-quality-sensor, and status-LED layers are present.	250	equal

Table 5.6: Results for the calculated variant-specific changes and changed delta operations

5.3 Variant Change Descriptor

This chapter evaluates the technical applicability of the *variant change descriptor*. The question is whether the description model of SGE can be applied to variant-specific changes that result from higher-order delta evolution. The existing description approach was originally introduced for describing changes in single-domain, non-variable systems. In this thesis, it is applied in a delta-oriented product-line setting. Therefore, the higher-order delta evolution must first be broke down to changes between the base and evolved model variants. These variant-specific changes are then described by applying the existing composed mapping.

The evaluation uses the same five model variant pairs as Section 5.2. For each pair, the *variant change descriptor* quantifies the share of changed modelling elements in the evolved model variant and classifies the changed operations into AVs and PVs. We compare the calculated descriptors with the manually derived ground truth. In addition, we compare raw and reduced calculations to analyse the influence of redundant delta operations.

5.3.1 Setup

We apply the `VariantChangeDescriptor` to each base/evolved model variant repository pair. First, `DeltaRepositoryDiff` calculates the variant-specific change delta repository that transforms the base model variant into the evolved model variant. Then, the descriptor counts the changed modelling elements in this diff and divides them by the number of modelling elements in the evolved variant repository. In this implementation, modelling elements are represented by delta operations. Technical root-initialisation operations and remove/destroy operations are excluded from the count. The counted changed operations are also classified using as AV/PV using the composed mapping. Thus, the descriptor reports both the share of changed modelling elements and the distribution of these changes over AVs and PVs.

We execute the evaluation in two modes. In raw mode, we use the exported repositories without removing redundant operations. In reduced mode, we first apply `DeltaRepositoryReducer` to the base and evolved model variant repositories. This removes redundant operations, such as overwritten modifications and add-remove pairs. The reduced mode is the main comparison to the ground truth because the ground truth describes the effective variant-specific change rather than redundant intermediate operations.

5.3.2 Ground Truth

Table 5.7 shows the manually derived ground truth for the reduced model variant repositories. The column *changed operations* corresponds to the changed modelling elements in the diff, while el_{m_i} denotes the counted modelling elements in the evolved model variant. The AV and PV columns show the variation distribution of the counted changed operations.

5.3.3 Results

The reduced calculation produces the same values as the ground truth for all five model variant pairs. The raw calculation differs slightly, which confirms the expectation that redundant delta operations influence the descriptor if they are not

Variant pair	Changed operations	el_{m_i}	$SoCME_{m_i}$	AV	PV
MV_1	232	233	0.9957	66.38%	33.62%
MV_2	187	391	0.4783	63.10%	36.90%
MV_3	471	833	0.5654	63.06%	36.94%
MV_4	266	1070	0.2486	57.89%	42.11%
MV_5	250	1273	0.1964	56.40%	43.60%

Table 5.7: Ground truth for the variant change descriptor

Variant pair	Raw			Reduced		
	changed	el_{m_i}	$SoCME_{m_i}$	changed	el_{m_i}	$SoCME_{m_i}$
MV_1	235	236	0.9958	232	233	0.9957
MV_2	185	394	0.4695	187	391	0.4783
MV_3	480	847	0.5667	471	833	0.5654
MV_4	264	1084	0.2435	266	1070	0.2486
MV_5	245	1302	0.1882	250	1273	0.1964

Table 5.8: Calculated variant change descriptors

removed before the calculation. Table 5.8 shows both results. Table 5.9 additionally shows the AV/PV distributions for the raw and reduced calculations.

The highest share is measured for MV_1, because the base variant contains only the core behaviour, while the evolved variant already contains wiper, low-quality sensor, and air-conditioning behaviour. The share decreases for MV_4 and MV_5 because these variants already contain a large amount of unchanged behaviour. MV_3 is an exception to the decreasing trend, because the automatic-headlights layer contributes a comparatively large number of changed operations. Across all variants, the AV share is higher than the PV share.

Variant pair	Raw		Reduced	
	AV	**PV**	**AV**	**PV**
MV_1	66.81%	33.19%	66.38%	33.62%
MV_2	62.70%	37.30%	63.10%	36.90%
MV_3	63.75%	36.25%	63.06%	36.94%
MV_4	57.58%	42.42%	57.89%	42.11%
MV_5	57.14%	42.86%	56.40%	43.60%

Table 5.9: AV/PV distributions of the calculated variant change descriptors

5.4 Aggregated Variant Change Descriptor

This chapter evaluates SG2a. The goal is to evaluate the technical applicability of the *variant space change descriptor* by assessing whether the *variant space change descriptor* can be derived by aggregating the *variant change descriptors* of the considered model variants. The aggregation is evaluated for the model variants of the subject system. We derive the expected *variant space change descriptor* manually from the reduced ground-truth values of Chapter 5.3 and compare it with the calculated aggregation to prove that the aggregation of the change descriptors is technical applicability.

5.4.1 Setup

We apply the `AggregatedVariantChangeDescriptor` to the five exported base/evolved variant repositories. For each model variant, the calculation first computes the same intermediate values as the `VariantChangeDescriptor`. Afterwards, it calculates the share of changed modelling elements by summing all changed operations and dividing this sum by the total number of counted operations in the evolved model variant repositories. Thus, the aggregation weights each model variant by its number of evolved modelling elements.

The implementation also aggregates the AV/PV counts of all model variant diffs. It reports both their contribution to $SoCME_{MS}$ and their relative distribution within the changed operations. As in Chapter 5.3, the main evaluation uses reduced variants because the reduced calculation describes the effective changes. The raw calculation is also reported to make the influence of redundant operations visible.

5.4.2 Ground Truth

The ground truth for the *variant space change descriptor* can be seen in the reduced ground truth values in Table 5.7. The sum of changed operations is $el_F = 1406$, consisting of $el_F^{AV} = 864$ AV operations and $el_F^{PV} = 542$ PV operations. The total number of counted modelling elements in the evolved variant repositories is $el_{150} = 3800$. Consequently, the expected value for the model variant space is:

$$SoCME_{MS} = \frac{el_F}{el_{150}} = \frac{1406}{3800} = 0.37$$

The expected AV and PV contributions are:

$$SoCME_{MS}^{AV} = \frac{864}{3800} = 0.2274, \qquad SoCME_{MS}^{PV} = \frac{542}{3800} = 0.1426.$$

This corresponds to an AV/PV distribution of 61.45% AV and 38.55% PV among the changed operations.

5.4.3 Results

The reduced aggregation matches the manually derived ground truth. This shows that the individual *variant change descriptors* can be aggregated into a descriptor for the evaluated model variant space. The raw aggregation produces a slightly lower value because the unreduced repositories contain a different number of redundant operations in the individual variants. Table 5.10 shows the results, including the AV/PV decomposition.

Calculation	el_F	el_{150}	$SoCME_{MS}$	$SoCME^{AV}_{MS}$	$SoCME^{PV}_{MS}$	AV/PV
Ground truth	1406	3800	0.3700	0.2274	0.1426	61.45% / 38.55%
Raw variants	1409	3863	0.3647	0.2255	0.1393	61.82% / 38.18%
Reduced variants	1406	3800	0.3700	0.2274	0.1426	61.45% / 38.55%

Table 5.10: Results for the aggregated variant change descriptor with AV/PV decomposition

5.5 Variant Space Change Descriptor

This chapter evaluates SG2b. The goal is to evaluate the feasibility of directly calculating the *variant space change descriptor* by determining whether the *variant space change descriptor* can be calculated directly from the complete base and evolved delta model without relying on individual model variants as input. We use the same delta model from the subject system as in the previous evaluations to ensure that the same model variants are considered as in Chapter 5.4.

The direct calculation is evaluated by comparison with the reduced aggregation-based calculation from Section 5.4. The relevant question is whether the descriptor values for the same evaluated model variant space are produced by both calculation paths. This answers the question of whether the redundant delta operations were taken into account in the same way, and whether the weighting of each model variant has the same impact as in the aggregation-based path.

5.5.1 Setup

We apply the `VariantSpaceChangeDescriptor` to the complete base repository, the complete evolved repository and the explicit variant definition file. The calculation builds a feature-selection derivation graph for both repositories, closes each base and evolved feature selection, selects the active deltas for each variant, reduces the resulting model variant repositories and computes the model variant-specific diff. It then aggregates the changed operations and evolved operations in the same way as the aggregation-based calculation from Section 5.4.

The AV/PV impact distribution is calculated for each model variant diff and then accumulated to obtain $SoCME^{AV}_{MS}$, $SoCME^{PV}_{MS}$ and the overall AV/PV split. This setup evaluates whether the direct calculation can reproduce the same descriptor as the aggregation over reduced model variant repositories.

5.5.2 Ground Truth

The ground truth is the same as for SG2a, because both approaches are expected to describe the same model variant space. Therefore, the direct calculation is expected to produce $el_F = 1406$, $el_{150} = 3800$, and $SoCME_{MS} = 0.37$. It is also expected to produce $SoCME^{AV}_{MS} = 0.2274$, $SoCME^{PV}_{MS} = 0.1426$, and an AV/PV distribution of 61.45% AV and 38.55% PV.

5.5.3 Results

The direct calculation produces exactly the same per-variant and aggregated values as the aggregation over reduced variant repositories. Table 5.11 shows the per-variant contributions calculated by the direct approach, including the AV/PV distribution.

Variant pair	Changed operations	el_{m_i}	AV	PV
MV_1	232	233	66.38%	33.62%
MV_2	187	391	63.10%	36.90%
MV_3	471	833	63.06%	36.94%
MV_4	266	1070	57.89%	42.11%
MV_5	250	1273	56.40%	43.60%
Total	**1406**	**3800**	**61.45%**	**38.55%**

Table 5.11: Directly calculated variant-space contributions with AV/PV distribution

The resulting descriptor is $SoCME_{MS} = 0.37$, with $SoCME^{AV}_{MS} = 0.2274$ and $SoCME^{PV}_{MS} = 0.1426$. This equals the reduced aggregated result from Chapter 5.4. Thus, for the subject system, the direct *variant space change descriptor* calculation is feasible and produces the same result as the aggregation of individual variant change descriptors, including the AV/PV decomposition.

5.6 Discussion & Threats to Validity

Discussion

The evaluation results address the four evaluation aspects defined at the beginning of this chapter.

First, the derivation of variant-specific changes from higher-order delta evolution is correct for the selected subject system. For all five evaluated model variant pairs, the calculated variant-specific changes match the manually derived ground truth. This includes both the expected feature layers and the number of changed delta operations. The result is important because the variant-specific change is the basis for all following descriptor calculations. If the diff between base and evolved model variants did not represent the expected evolution, the calculated descriptors would measure the wrong change.

Second, the evaluation shows that the *variant change descriptor* is technically applicable in the context of this thesis. The existing description approach can be applied after breaking down higher-order delta evolution to variant-specific changes. The resulting descriptor quantifies the share of changed modelling elements for each model variant pair and classifies the changed operations into AV and PV. The reduced calculation matches the manually derived ground truth for all five pairs. This indicates that the mapping to the description model of SGE can be used not only for single-domain, non-variable systems, but also for describing variant-specific changes in a delta-oriented product-line setting.

The values of the *variant change descriptor* also provide meaningful information about the evaluated evolution. MV_1 has the highest change share with $SoCME_{m_i} = 0.9957$, because the base variant contains only the core behaviour, while the evolved variant already contains wiper, low-quality-sensor and air-conditioning behaviour. MV_4 and MV_5 have lower change shares because these variants already contain a larger amount of unchanged behaviour. MV_3 does not follow a strictly decreasing trend because the automatic-headlights layer introduces many additional state-machine operations. The AV/PV distribution shows that AVs dominate the evaluated changes. However, PVs still account for a substantial share of the change, ranging from 33.62% to 43.60% in the reduced variant-specific results. This means that the evolution is not limited to changing attribute values, but also affects structural modelling principles.

Third, the aggregation-based *variant space change descriptor* is technically applicable for the evaluated model variant space. Aggregating the reduced *variant change descriptors* yields $SoCME_{MS} = 0.37$, which matches the manually derived ground truth. This result shows that the changes of individual model variants can be combined into one descriptor for the evaluated model variant space. The descriptor also preserves the AV/PV decomposition. In the evaluated subject system, the changed operations consist of 61.45% AV and 38.55% PV. This provides a compact summary of the evolution of the evaluated model variant space.

Fourth, the direct *variant space change descriptor* calculation is feasible for the subject system. The direct calculation produces the same per-variant contributions and the same aggregated values as the aggregation over reduced model variant repositories. This shows that the direct calculation can reproduce the same descriptor without requiring the exported model variant repositories as explicit input. For the evaluated subject system, both calculation paths therefore describe the same model variant space consistently.

The evaluation also shows why reduction is relevant. Raw and reduced calculations are not identical. For example, the raw aggregation yields $SoCME_{MS} = 0.3647$, while the reduced aggregation yields $SoCME_{MS} = 0.37$. The AV/PV distribution changes from 61.82% AV and 38.18% PV in the raw aggregation to 61.45% AV and 38.55% PV after reduction. This confirms that redundant delta operations can affect the descriptor, even when they do not contribute to the effective model variant-level change. Therefore, the reduced calculation is better suited for evaluating the effective change of the evolved PL.

In summary, the evaluation provides evidence that the implementation derives the expected variant-specific changes, that the *variant change descriptor* can be applied to these changes, that the aggregation-based *variant space change descriptor* correctly summarises the evaluated model variant space, and that the direct calculation is a feasible alternative calculation path for the same descriptor.

Threats to validity

The main threats to construct validity concern the definition of modelling elements and the interpretation of the resulting descriptor values. In this thesis, modelling elements are represented by delta operations. Technical root-initialisation operations and remove/destroy operations are excluded from the count. This definition

is consistent with the implemented descriptor and with the purpose of measuring structural change in the delta representation. However, a different definition of modelling elements could lead to different numeric values.

A second construct threat concerns the AV/PV classification. The classification depends on the mapping from delta operation types to AVs and PVs. This mapping makes the descriptor interpretable in terms of the description model of SGE, but it also abstracts from the detailed modelling context in which an operation occurs. Therefore, the AV/PV distribution should be interpreted as a classification of changed delta operations, not as a complete semantic classification of the evolved system behaviour.

A third construct threat is that the descriptor measures structural change in the delta representation rather than behavioural equivalence of the resulting executable system. Two different delta level changes may lead to behaviourally equivalent variants, while small delta level changes may have large behavioural consequences. The evaluation therefore supports the structural description of model variant and model variant space evolution, it does not describe the effects of evolved BCS variants' behaviour.

Threats to internal validity mainly concern the manually derived ground truth and the implementation of the evaluation infrastructure. The ground truth was derived manually from the exported base and evolved variant repositories. Manual derivation can introduce mistakes, especially when feature dependencies and redundant operations are involved. To mitigate this threat, the evaluation separates the expected feature-level changes, the changed operation counts and the descriptor values into several tables, so that inconsistencies become easier to identify. The implementation is another internal threat because the calculation depends on several components, including feature-selection closure, repository export, repository reduction, diff calculation and descriptor calculation. An error in one of these components could influence the results. This threat is reduced by comparing several intermediate results with the ground truth, rather than only comparing the final aggregated descriptor.

Threats to external validity concern the generalisability of the results. The evaluation uses one subject system based on the BCS and one tailored evolution scenario. Although the BCS is an established and realistic academic case study, the evaluated subject system covers only a state-machine-based delta model and five explicitly selected model variant pairs. Therefore, the evaluation does not show that the same results hold for all domains, all delta dialects or all possible product-line evolution scenarios. In particular, the direct calculation of the *variant space change descriptor* should also be evaluated on larger subject systems with more variants and different evolution patterns. Such evaluations are necessary to assess how well the approach generalises beyond the selected BCS-based subject system.

5.7 Summary

This chapter evaluated the implemented change descriptors on a BCS-based subject system. The subject system consists of a complete base delta repository, a complete evolved delta repository, five evaluated base/evolved model variant pairs, and the newly introduced `AirConditioning` evolution delta.

The first evaluation aspect showed that the implementation derives the expected variant-specific changes from higher-order delta evolution for all five evaluated model variant pairs. This confirms the required basis for the following descriptor calculations.

The second evaluation aspect showed that the *variant change descriptor* is technically applicable to the variant-specific changes. The reduced descriptors match the manually derived ground truth and classify the changed operations into AV and PV, thereby applying the description model of SGE in the delta-oriented product-line context of this thesis.

The third evaluation aspect showed that the *variant space change descriptor* can be calculated by aggregating the reduced *variant change descriptors*. The aggregation yields $SoCME_{MS} = 0.37$, with $SoCME_{MS}^{AV} = 0.2274$ and $SoCME_{MS}^{PV} = 0.1426$. The changed operations are distributed as 61.45% AV and 38.55% PV.

The fourth evaluation aspect showed that the direct *variant space change descriptor* calculation produces the same result as the aggregation-based calculation for the evaluated subject system. This demonstrates the feasibility of the direct calculation path for the selected model variant space.

Overall, the evaluation provides evidence that the implemented approach can describe the evolution of individual model variants and the evaluated model variant space for the selected subject system. The main limitations are the restricted number of model variant pairs, the focus on one state-machine-based BCS delta model, the manual derivation of the ground truth, and the limited generalisability to other domains and larger PLs.

6. Related Work

This chapter discusses research that is related to the central concerns of this thesis: the evolution of SPLs, the management of variability over time, and the description and reasoning of changes to model variants and model variant spaces. Since this thesis builds on delta modelling as a mechanism for defining and evolving model variants, particular attention is given to approaches that either extend delta modelling itself or address comparable evolution problems from a different perspective.

6.1 Higher-Order Delta Modelling for Software Product Line Evolution

Lity et al. [10] introduce higher-order delta modelling as a construct to evolve existing delta models, thereby enabling the evolution of PLs using delta modelling. We have already discussed this work extensively in Chapters 2.3 and 3.1.2, as it forms one of the foundations of this thesis. However, we have not examined the section in which the reasoning behind the application of higher-order delta is explained. In this section, the evolution of the model variant space and the individual model variants is described mathematically, thereby defining the changes to both depending on the higher-order delta. While this theoretical and mathematical definition can be seen as a parallel approach to defining the same changes to the model variant space and individual model variants as in our thesis, it is unsuitable for practical use in real-world scenarios.

6.2 Sometimes You Have to Treat the Symptoms: Tackling Model Drift in an Industrial Clone-and-Own Software Product Line

Tinnes et al. [19] present an empirical industrial case study that addresses the phenomenon of 'model drift' in SPLs that are managed using a clone-and-own approach. The paper details how independent modifications to cloned variants cause them to diverge over time and proposes practical, symptom-focused solutions to realign and

consolidate these models within a more managed lifecycle. This research provides real-world evidence to support our thesis, demonstrating the significant maintenance issues that arise when the evolution of model variants is not monitored or tracked. Their approach can be used as a diagnostic toolkit to rescue and prepare legacy variants that have drifted for migration into a structured PL. While our thesis builds upon this problem space, it also offers a proactive, preventive alternative. Rather than treating the symptoms of drift via clone analysis after the fact, our work utilises delta models to explicitly define, describe and govern the evolution of variants and the variant space by design. This helps to prevent unmanaged drift from occurring in the first place.

6.3 Semantic Evolution Analysis of Feature Models

The paper by Drave et al. [7] focuses on analysing the semantic evolution of feature models within SPLs. The authors present a formal framework for analysing the effect of changes to a feature model on its semantic meaning, and specifically how the set of valid configurations shrinks, grows or shifts over time. Product line engineers can use this approach to automatically detect whether a modification introduces breaking changes, preserves backward compatibility, or creates entirely new valid product spaces. In the context of our thesis, this paper provides a foundational understanding of model variant space evolution at the abstract variability level. However, whereas Drave et al. operate strictly within the high-level configuration space of feature models, our work directly links this to the concrete solution space by mapping these changes onto the evolution of model variants and the underlying delta models. Therefore, their semantic analysis serves as a complementary mechanism for validating the model variant spaces we are evolving and describing.

6.4 Reasoning about product-line evolution using complex feature model differences

Bürdek et al. [4] introduce a structured approach to reasoning about product-line evolution, involving the calculation and analysis of complex differences between feature model revisions. Rather than relying on low-level, atomic edit operations, their framework elevates differences to a higher semantic level, identifying complex refactorings, generalisations or specialisations of the configuration space. This methodology is valuable for understanding how a PL's configuration capabilities change across different versions, enabling developers to generate precise migration paths for existing products. While our thesis shares the core objective of capturing structural and semantic changes over time, it shifts the operational paradigm from state-based feature model differences to constructive delta-oriented modelling. While Bürdek et al. analyse the evolution of the model variant space via a comparative 'before-and-after' analysis of the variability model, our approach utilises delta models and the description model of SGE to express the differences between individual model variants and their evolution within the PL.

7. Conclusion and Outlook

This thesis addressed the question of how the evolution of delta-oriented PLs can be described in a structured and comprehensible way. In particular, it focused on describing the effects of higher-order delta evolution not only for individual model variants, but also for the complete model variant space.

We formalised how the evolution of an individual model variant can be described using the description model of SGE. For this purpose, we defined the *variant change descriptor*, which captures the changes introduced to a specific model variant during evolution. This enables the effort and risk caused by the evolution of individual model variants to be estimated more systematically. Building on this, we extended the perspective from individual model variants to the whole model variant space and defined the *variant space change descriptor*. This descriptor makes it possible to describe how the model variant space evolves as a result of higher-order delta evolution.

In addition to these conceptual contributions, we introduced structural foundations and algorithms for calculating the proposed descriptors. The 150% model, delta derivation trees and split-induced subtree forests provide the basis for representing and calculating the relevant relations between delta models, higher-order deltas and model variants. Based on these structures, two approaches for calculating the *variant space change descriptor* were presented. The first approach aggregates the *variant change descriptor* of all individual model variants. The second approach calculates the *variant space change descriptor* directly and is expected to be more efficient for large delta models, especially when only a small part of the delta model is affected by evolution.

We evaluated the proposed concepts by applying the implemented change descriptor calculations to the adapted BCS subject system. The evaluation confirmed the correctness of the calculation for variant-specific changes. Furthermore, it demonstrated the technical applicability of both the *variant change descriptor* and the aggregation-based calculation of the *variant space change descriptor*. The direct calculation of the *variant space change descriptor* produced the same results, which demonstrates the feasibility of this alternative calculation path.

However, it remains future work to prove that the direct calculation of the *variant space change descriptor* is more efficient than the aggregation-based approach. Such an evaluation would require a subject system with a large number of model variants in which only a comparatively small part of the delta model evolves. This subject system would also need to be implemented as a delta repository using the delta dialect. Depending on the domain of the subject system, only minor implementation adjustments may be necessary to apply the concepts developed in this thesis.

The implementations created for this thesis were primarily designed for evaluating the proposed concepts using the BCS subject system. Therefore, they are currently tailored to this specific subject system. Since the design of this thesis is independent of a particular delta model domain, further work could adapt the implementations for more general use. This would make it possible to apply the proposed descriptors to other delta repositories and thereby support the analysis of product-line evolution in a broader range of domains.

A further direction for future work concerns the extension of the proposed concepts to multi-domain PLs. In this thesis, the calculations were applied to a delta model from a single domain. However, product-line evolution often affects several related domains in parallel. Future work could therefore investigate whether the descriptors introduced in this thesis can be extended to a multi-domain variant space change descriptor. Such a descriptor would not merely aggregate more evaluation data, but would describe how changes in different domains relate to each other during product-line evolution.

This could enable a more comprehensive understanding of product-line evolution. For example, it could help to analyse whether a change in one domain is reflected consistently in other domains, or whether evolution introduces mismatches between related artefacts. In this way, the concepts developed in this thesis could serve as a foundation for reasoning not only about the evolution of individual model variants and model variant spaces, but also about the coordinated evolution of complete multi-domain PLs.

Bibliography

[1] Albert Albers and Simon Rapp. "Model of SGE: system generation engineering as basis for structured planning and management of development". In: *Design Methodology for Future Products: Data Driven, Agile and Flexible*. Springer, 2021, pp. 27–46.

[2] Sven Apel et al. "Software product lines". In: *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013, pp. 3–15.

[3] Steffen Becker et al. "A process model and classification scheme for semi-automatic meta-model evolution". In: *1st Workshop MDD, SOA und IT-Management (MSI), GI, GiTO-Verlag*. 2007, pp. 35–46.

[4] Johannes Bürdek et al. "Reasoning about product-line evolution using complex feature model differences". In: *Automated Software Engineering* 23.4 (2016), pp. 687–733.

[5] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. "Abstract delta modeling". In: *ACM Sigplan Notices* 46.2 (2010), pp. 13–22.

[6] Ferruccio Damiani and Michael Lienhardt. "Refactoring delta-oriented product lines to achieve monotonicity". In: *arXiv preprint arXiv:1604.00346* (2016).

[7] Imke Drave et al. "Semantic evolution analysis of feature models". In: *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*. 2019, pp. 245–255.

[8] Arne Haber et al. "Evolving delta-oriented software product line architectures". In: *Monterey Workshop*. Springer. 2012, pp. 183–208.

[9] Jörg Christian Kirchhof et al. "Variant and product line co-evolution". In: *Model-Based Engineering of Collaborative Embedded Systems: Extensions of the SPES Methodology*. Springer, 2020, pp. 333–351.

[10] Sascha Lity, Matthias Kowal, and Ina Schaefer. "Higher-order delta modeling for software product line evolution". In: *Proceedings of the 7th International Workshop on Feature-Oriented Software Development*. 2016, pp. 39–48.

[11] Sascha Lity et al. "Delta-oriented software product line test models-the body comfort system case study". In: *Technical report, TU Braunschweig* (2013).

[12] Tobias Müller et al. "A comprehensive Description of a Model-based, continuous Development Process for AUTOSAR Systems with integrated Quality Assurance". In: *Informatik-Bericht. TU Braunschweig* (2009).

[13] Sophia Nahrendorf, Sascha Lity, and Ina Schaefer. “Applying higher-order delta modeling for the evolution of delta-oriented software product lines”. In: *TU Braunschweig-Institute of Software Engineering and Automotive Informatics; Technical Report; Institute for Software Engineering and Automotive Informatics: Braunschweig, Germany* (2018).

[14] Dirk Neumann, Tobias Pett, and Ina Schaefer. “Towards Consistency Preservation for Multi-Domain Variability Modeling”. In: *Proceedings of the 2025 29th ACM International Systems and Software Product Line Conference-Volume A*. 2025, pp. 88–93.

[15] Ochs et al. “Describing Changes in Multi-Disciplinary Engineering”. Work-in-Progress.

[16] Ina Schaefer. “Variability modelling for model-driven development of software product lines.” In: *VaMoS* 10 (2010), pp. 85–92.

[17] Ina Schaefer et al. “Software diversity: state of the art and perspectives”. In: *International Journal on Software Tools for Technology Transfer* 14.5 (2012), pp. 477–495.

[18] Sandro Schulze, Oliver Richers, and Ina Schaefer. “Refactoring delta-oriented software product lines”. In: *Proceedings of the 12th annual international conference on Aspect-oriented software development*. 2013, pp. 73–84.

[19] Christof Tinnes et al. “Sometimes you have to treat the symptoms: tackling model drift in an industrial clone-and-own software product line”. In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2022, pp. 1355–1366.

A. Appendix

Figure A.1: AirConditioning Delta

```
# AirConditioning Evaluation

Feature selections and repository pairs for the AirConditioning evaluation

The evaluation compares base and evolved delta model repositories to calculate the  variant change descriptor.

Addtionally, the evaluation compares two equivalent ways to calculate the variant space change descriptor:

1. VariantSpaceChangeDescriptor starts from the complete base and evolved delta model repositories, selects each active base/evolved   variant by feature selection, reduces the selected variant-local repositories, and then calculates the descriptor.

2. AggregatedVariantChangeDescriptor in REDUCED_VARIANTS mode starts from already existing base/evolved variant repositories, reduces each repository, and then calculates the descriptor.


Generation rule
The V*_base.genericdeltametamodel and V*_evolved.genericdeltametamodel files in this directory are created with VariantEvaluationRepositoryExporter. The exporter uses the same FeatureSelectionDerivationGraph logic as VariantSpaceChangeDescriptor.

This is important because the graph closes explicit feature selections with derived features before it selects active deltas. For the current air-conditioning repository, the derived features are:
RemoteControllKey
LowQualitySensor
LEDClean
LEDWiper

Variant feature-pair definitions for VariantSpaceChangeDescriptor
V1_add_wiper_ac: base=[] -> evolved=[Wiper, AirConditioning]
V2_add_seat: base=[Wiper] -> evolved=[Wiper, AirConditioning, Seat]
V3_add_automatic_headlights: base=[Wiper, Seat] -> evolved=[Wiper, AirConditioning, Seat, AutomaticHeadlights]
V4_add_clean_manpw: base=[Wiper, Seat, AutomaticHeadlights] -> evolved=[Wiper, AirConditioning, Seat, AutomaticHeadlights, Clean, ManualPowerWindow]
V5_add_hqs_statusled: base=[Wiper, Seat, AutomaticHeadlights, Clean, ManualPowerWindow] -> evolved=[Wiper, AirConditioning, Seat, AutomaticHeadlights, Clean, ManualPowerWindow, HighQualitySensor, StatusLED]
```

Listing A.1: Markdown Evaluation