Lifting Multi-Domain Consistency Preservation to Variant-Rich Systems

Simon Bothe
Karlsruhe Institute of Technology
Karlsruhe, Germany
simon.bothe@kit.edu

Dirk Neumann
Karlsruhe Institute of Technology
Karlsruhe, Germany
dirk.neumann@kit.edu

Tobias Pett
Karlsruhe Institute of Technology
Karlsruhe, Germany
tobias.pett@kit.edu

Ina Schaefer
Karlsruhe Institute of Technology
Karlsruhe, Germany
ina.schaefer@kit.edu

Abstract

Modern software-intensive systems require models of different engineering domains, such as electronics, mechanics, and software. Keeping these models consistent across domains is essential for designing a valid system. Software-intensive systems tend to be variant-rich because of market demands for individualization. The state of the art for consistency preservation in variant-rich systems is modeling each variant individually and applying consistency preservation mechanisms to them. This approach leads to redundant modeling of model elements shared between variants and does not scale for large systems. To solve this issue, we present a novel approach to combine multi-domain consistency preservation with delta-oriented variability modeling. By lifting the Vitruvius consistency preservation mechanism to delta models, we can preserve consistency for all variants simultaneously, avoid redundant modeling, and enable scalable consistency preservation for variant-rich multi-domain systems. Our evaluation shows this approach preserves consistency for all system variants and scales well enough to be applicable to large systems.

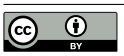
CCS Concepts

• **Software and its engineering** → **Software product lines**; **Consistency**.

Keywords

Consistency Preservation, Variability Modeling, Multi-Domain Variability

ACM Reference Format:
Simon Bothe, Dirk Neumann, Tobias Pett, and Ina Schaefer. 2026. Lifting Multi-Domain Consistency Preservation to Variant-Rich Systems. In *ACM/IEEE 29th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion 2026), October 04–09, 2026, Málaga, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3837062.3839063



This work is licensed under a Creative Commons Attribution 4.0 International License.
MODELS Companion 2026, Málaga, Spain
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2903-4/26/10
https://doi.org/10.1145/3837062.3839063

1 Introduction

The engineering of modern software-intensive systems, such as automotive systems, requires knowledge across multiple *domains* [23]. A domain describes a bounded field of interest or knowledge [8], such as electronics, mechanics, and software. Specialized domain experts develop each system domain. The parallel development of system domains by their respective domain experts can lead to inconsistencies, which prevents designing a valid system. Inconsistencies between system domains mean the system domains do not fit together. An example would be the existence of a control unit behavior in software, while the electronic wiring plan does not include that control unit. There are several approaches to preserve such consistencies between system domain. One of these approaches is *Vitruvius* [15], which uses imperative consistency preservation rules. Consistency preservation rules react to a model change in a source domain with a consequential change in a target domain. Vitruvius can solve the example inconsistency by reacting to the addition of a control unit behavior in software, with the addition of that control unit to the electronic wiring plan. Modern software-intensive systems, such as automotive systems, tend to be variant-rich because of market demands [23]. Applying a consistency preservation approach like Vitruvius to variant-rich systems forces the usage of cloning to realize system variants since the consistency preservation rules are not variability-aware. This results in duplicates of model elements shared between variants and is generally considered a harmful practice [13].

To address this issue, we present a novel approach to combine the multi-domain consistency preservation of Vitruvius with delta-oriented variability modeling [25] [9]. A preliminary version of this approach has been presented in [7]. Delta-oriented variability modeling represents variant-rich systems by explicitly modeling the differences between variants in the form of deltas and ensuring model elements shared between variants only exist once. Existing consistency preservation rules are not variability-aware and need to be adapted by lifting them to deltas. Variability-aware consistency preservation rules for deltas can preserve consistency for all system variants simultaneously, avoid redundant modeling, and achieve scalability for large systems. In this paper, we describe a method to systematically lift consistency preservation rules to delta-oriented variability modeling and apply it to consistency preservation for the Body Comfort System case study [18]. We evaluate the lifting process by checking the consistency of system variants

constructed from consequential changes of the variability-aware consistency preservation approach and also evaluate the scalability of variability-aware consistency preservation by measuring the size of consequential changes. The evaluation results show consistency is preserved for all system variants, and applying this approach to large systems does not lead to significant scalability issues.

2 Background

Multi-domain Modeling and Consistency Preservation. Model-driven engineering can represent a system in views of different fields of interest or knowledge to separate concerns called domains [8]. Each domain consists of one domain meta-model, which describes the entities and relations between these entities in the domain, and instantiations of meta-models, called models. Modern software-intensive systems consist of multiple domains, such as electronics, mechanics, and software. The models describing such a multi-domain system often have semantic overlap, since they describe the same underlying system. If the overlapping model elements differ between the models, they are inconsistent, and ambiguities arise. Consistency preservation keeps the overlapping model elements synchronized between models to avoid ambiguities.

The *Vitruvius*-Approach [15] introduces the *Virtual Single Underlying Model* (V-SUM) to preserve consistency between different domain models while keeping their meta-models separated. A V-SUM is the instantiation of a *Virtual Single Underlying Meta-Model* (V-SUMM), which consists of domain meta-models and unidirectional *Consistency Preservation Rules* (CPRs) between them, that are manually defined between domain meta-models. Instances of these CPRs are responsible for preserving consistency defined by *Consistency Rules* (CRs) between a source and target domain model. A CR establishes which model elements have to be synchronized between the domain models. Whenever a change in the source domain model occurs that breaks a CR, the CPR applies *consequential changes* to the target domain model in order for the CR to hold again. The source and target domain models can be identical (*intra-domain consistency preservation*) or different (*cross-domain consistency preservation*). We call CPRs for systems without variability *Product Consistency Preservation Rules* (Product-CPRs) to distinguish them from CPRs for variant-rich systems.

Running Example - The Body Comfort System. We use the *Body Comfort System* (BCS) case study [18] as a running example of a variant-rich multi-domain system and the subject system for the evaluation of this paper. This case study has already been widely applied to variability research and models *Electronic Control Units* (ECUs) in a vehicle with 11616 possible system variants in two different domains: The component domain represents a structural system view for the ECUs and models ECU components, their interfaces, and connectors between them. The statemachine domain represents a system behavior view and models the behavior of each ECU by a statemachine. Both domains describe the same BCS system and have semantic overlap, such as the existence of a behavioral statemachine in the statemachine domain for every ECU component in the component domain. This overlap necessitates consistency preservation.

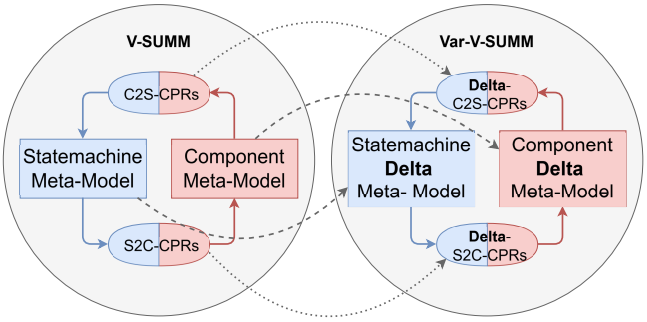


Figure 1: The lifting of a V-SUMM (left) to a Var-V-SUMM (right) for the BCS. The dashed arrows show the lifting of models to delta models, while the dotted arrows show the lifting of Product-CPRs to Delta-CPRs.

A V-SUMM for the BCS running example is depicted abstractly on the left side of Figure 1. This example V-SUMM includes domain models for the statemachine and component domain, as well as Product-CPRs *from the component to the statemachine domain model* (C2S) and *from the statemachine to the component domain model* (S2C). One of the possible C2S Product-CPRs can be seen in Listing 1. The Product-CPR is divided into a reaction (line 3), responsible for reacting to a change to the component domain model, and a routine (line 7) that gets called to preserve consistency by applying consequential changes to the statemachine domain model. The reaction includes the *trigger change*, which is the change for which the Product-CPR is executed (line 4). This example Product-CPR executes if an ECU component is added into the system in the component domain model. The routine includes the application of the consequential change (lines 13-14), as well as correspondence management. The consequential change in this example is the addition of a region to the statemachine. Correspondence management consists of retrieving (lines 9-10) and adding (line 15) correspondences from the *correspondence model*. The correspondence model stores relationships between model elements in different domains to track semantic overlap. In the example, a statemachine region in the statemachine domain model corresponds to a system component in the component domain model (lines 9 and 15) because each system component behavior must be described by a corresponding statemachine region.

Delta-oriented Variability Modeling. Delta-oriented variability modeling [25] [9] represents variant-rich systems by modeling variability in the form of change-capturing deltas. Deltas encapsulate delta operations, which are additions, removals, or changes of model entities in a system variant. By iteratively applying deltas and the contained delta operations, a system variant can be transformed into a different system variant. When choosing a common starting point for the transformations, each system variant can be expressed by a set of deltas, which result in that specific system variant when applied [26]. In order to decide when a delta is applied, an application condition and an application order can be specified. The application order defines an order in which deltas can be applied. This is important if deltas depend on each other.

```
import "statemachine_meta-model.ecore" as STMA
import "component_meta-model.ecore" as COMP
reaction ComponentAdded {
  after adding to COMP::System[components]
  call addRegion(newValue)
}
routine addRegion(COMP::Component comp) {
  match {
    requires absence of STMA::Region corresponding to
     comp
    val statemachine = retrieve STMA::Statemachine
     corresponding to getContainer(component)
  }
  update {
    val region = new STMA::Region()
    statemachine.regions.add(region)
    create correspondence between region and comp
} }
```

Listing 1: An example C2S Product-CPR. This Product-CPR executes with the addition of a component and reacts with the addition of a statemachine region as the consequential change.

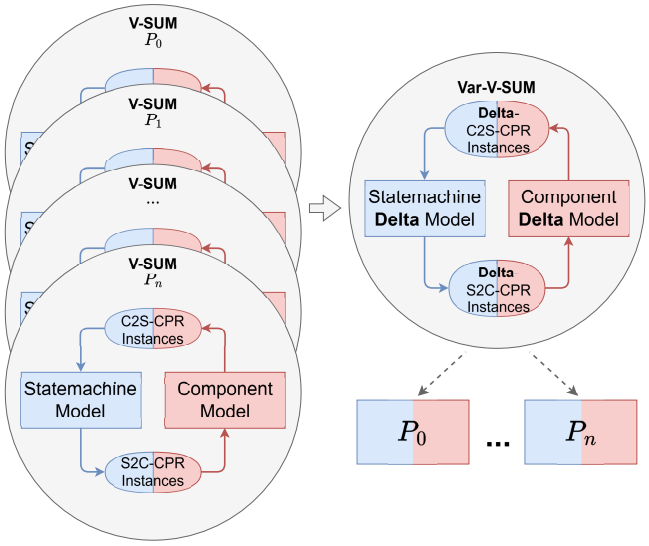


Figure 2: All system variant V-SUMs can be combined into a single Var-V-SUM. System variants can be derived from that Var-V-SUM and consistency is simultaneously preserved for all system variants by the Delta-CPRs.

DeltaEcore [28] can be used to generate a delta domain meta-model for any provided domain meta-model. By combining the generic DeltaEcore meta-model, which describes all generic delta operations, together with a selected domain meta-model, we generate a delta dialect for that domain meta-model. The resulting dialect is a domain meta-model itself, which describes all possible delta operations for the domain. For our BCS running example, we combine the generic DeltaEcore meta-model with the component domain meta-model, as well as the generic DeltaEcore meta-model with the statemachine domain meta-model, to generate delta dialects in the form of the statemachine delta domain meta-model and component delta domain meta-model. Instances of these meta-models allow delta operations to derive all possible BCS variants in the case study.

3 Extending Multi-Domain Consistency Preservation with Delta-oriented Variability

Applying the Vitruvius multi-domain consistency preservation mechanism to variant-rich multi-domain systems requires one V-SUM per variant. The V-SUMs include redundancies in the form of duplicated model elements, since they are shared by different variants of the system. These redundancies lead to additional maintenance, e.g., when evolving the system variants. We address redundancies with the introduction of a *Variability-aware Virtual Single Underlying Model* (Var-V-SUM) by applying delta-oriented variability modeling to facilitate model reuse between variants and by using *Delta Consistency Preservation Rules* (Delta-CPRs) to ensure consistency is preserved for all system variants. This represents a partial realization of the vision presented in Eger et al. [11].

3.1 Introducing Variability Awareness

A V-SUM consists of domain models to represent the system domains and CPR instances (specifically Product-CPR instances) to preserve consistency between the domain models. All semantic overlaps between the models get synchronized by Product-CPR applications whenever a change occurs. This makes the V-SUM appear as a single model from the outside while keeping the domain meta-models separated. A Var-V-SUM also appears as a single model from the outside but has the ability to include multiple system variants. The variability-aware CPRs (Delta-CPRs) synchronize semantic overlaps between the domains for all included variants. Migrating from V-SUMs to a Var-V-SUM requires changing the meta-model. A V-SUMM is transformed into a Var-V-SUMM by changing the domain meta-models to variability-aware delta domain meta-models and changing the Product-CPRs to variability-aware Delta-CPRs. We call these changes to achieve variability-awareness *liftings*.

Consider the liftings for the BCS running example case study in Figure 1. To lift the domain meta-models, we use DeltaEcore [28] to obtain delta domain meta-models for both the statemachine and component domain. The dashed arrows in the figure represent this lifting. Second, we lift Product-CPRs to Delta-CPRs and apply them between the created delta models in order to preserve consistency between the delta domains. The CPR lifting is required since Product-CPRs are not compatible with delta domain models. This lifting process (represented by the dotted arrows) is explained in Section 3.3.

Figure 2 depicts V-SUMs and the Var-V-SUM before and after the meta-model is lifted: Instead of having a dedicated V-SUM for each BCS variant (P_0 to P_n) to preserve consistency via Product-CPRs (left side), lifting their domain meta-models and Product-CPRs allows the derivation of all system variants from the single Var-V-SUM and the preservation of consistency between domains for all variants simultaneously via Delta-CPRs (right side). The usage of one Var-V-SUM for all system variants instead of one V-SUM per system variant eliminates redundancies.

```
1  import "delta_generic_meta-model.ecore" as GEN
2  import "delta_statemachine_meta-model.ecore" as DSTMA
3  import "delta_component_meta-model.ecore" as DCOMP
4  reaction ComponentAdditionAdded {
5    after adding DCOMP::AddComponentToSystemOp to GEN::
       Delta[operations]
6    call addRegionAddition(newValue)
7  }
8  routine addRegionAddition(DCOMP::AddComponentToSystemOp
       addCompOp) {
9    match {
10     requires absence of DSTMA::AddRegionOp corresponding
        to addCompOp
11     val addStatemachineOp = retrieve DSTMA::
        AddStatemachineOp corresponding to
        getContainerAddOperation(addCompOp)
12   }
13   update {
14     val addRegionOp = new DSTMA::AddRegionToSTMAOp()
15     addRegionOp.affectedObjectOp = addStatemachineOp
16     val mirrorDelta = getMirrorDeltaForOp(addCompOp)
17     mirrorDelta.addOperation(addRegionOp)
18     create correspondence between addRegionOp and
        addCompOp
19 } }
```

Listing 2: An example C2S Delta-CPR. This Delta-CPR triggers for the addition of a component addition operation and reacts with the addition of a statemachine region addition operation as the consequential change.

3.2 Delta Consistency Preservation Rules

Product-CPRs preserve consistency for a single multi-domain system variant. A Product-CPR enforces a CR by ensuring that this CR always holds. Delta-CPRs preserve consistency for all system variants and operate on deltas, which are used to derive these system variants. A Delta-CPR enforces a CR by ensuring that this CR always holds for all derivable system variants. Delta-CPRs react to changes to deltas with consequential changes to deltas. Since a delta can be applied for the derivation of multiple system variants, it allows the Delta-CPR to react to changes to multiple system variants with consequential changes to those system variants.

An example for a Delta-CPR can be seen in Listing 2. The CR enforced by this Delta-CPR is identical to the CR enforced by the Product-CPR in Listing 1. The trigger change of this Delta-CPR is the addition of a component addition operation to a delta (line 5). If this change is applied to the component delta model, the Delta-CPR is executed. This component addition represents the addition of an ECU, which can be present in multiple BCS variants. The consequential change is the addition of a statemachine region addition operation (lines 14-15) to a delta in the statemachine delta model, which is determined by a specific *localization strategy* (see below). In this case, the target delta is a mirrored delta of the source delta (lines 16-17). The new consequential change enforces the existence of a corresponding statemachine region for all system components in any derivable BCS variant (enforcement of the same CR as the Product-CPR presented in Listing 1). The Delta-CPR uses the same correspondence model as a Product-CPR for correspondence management but creates correspondences exclusively between delta operations (lines 10-11 and line 18).

Localization Strategies. Consequential changes of Delta-CPRs have to be placed in target domain deltas, which are applied to derive system variants (see Listing 2 lines 16-17). Delta-CPRs react to a change in a delta of the source domain with a consequential change in a delta of the target domain. We call the source domain delta the *change source delta* and the target domain delta the *change target delta* for a Delta-CPR application. We propose three strategies for the change target delta selection: Delta mirroring, functional delta assignment, and configuration-dependent assignment.

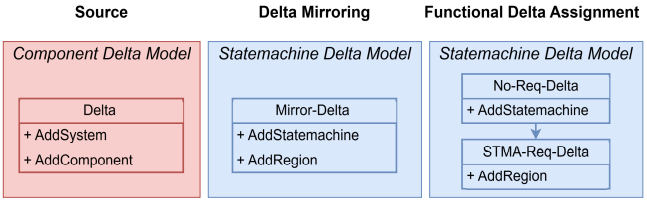


Figure 3: Localization strategies for Delta-CPRs: The addition of delta operations to the component delta model places consequential changes into different deltas in the statemachine delta model depending on the localization strategy. The arrow between deltas on the right side indicates the application order.

Delta mirroring can be used whenever the consequential change in the target domain model is independent of the system variant it is applied to. In these cases, the Delta-CPR applies the consequential changes to all variants that are affected by the Delta-CPR trigger change. Delta mirroring aims to preserve the grouping intent of the delta operations in the source domain model. Consequential changes are placed into the change target delta, which mirrors the change source delta. A delta mirroring example for cross-domain consistency is depicted in Figure 3. Adding the system and component addition operations into the component delta model on the left requires adding a statemachine and statemachine region addition operation into the statemachine delta model. Delta mirroring places the consequential delta operation in a statemachine delta in the middle, which mirrors the change source delta on the left.

Functional delta assignment can be used as an alternative to delta mirroring whenever the consequential change in the target domain model is independent of the system variant it is applied to. In these cases, the Delta-CPR applies the consequential changes to all variants that are affected by the Delta-CPR trigger change. Functional delta assignment aims to create delta dependency chains with clear and comprehensible application orders. It identifies the deltas in the target domain that are required to be applied before performing a consequential change. A valid application of the change target deltas must be ensured by setting their application order accordingly. A functional delta assignment example for cross-domain consistency is depicted in Figure 3. Adding the system and component addition operations into the component delta model on the left requires adding a statemachine and statemachine region addition operation into the statemachine delta model. The consequential delta operations are placed in individual change target deltas on the right, based on the deltas that are required to be applied before. Adding a statemachine does not require any prior delta operations. But adding a statemachine region requires the addition of a statemachine before, so the region addition operation has to be placed in a

change target delta, which is only applied after a statemachine has been added.

Configuration-dependent assignment must be used whenever the consequential change in the target domain model depends on specific system variants. In these cases, the Delta-CPR is required to apply the consequential changes only to a subset of variants that are affected by the Delta-CPR trigger change. A BCS example exists in the component domain model. If an input and an output port of the same signal exist in components, a connector is created between them. While a Product-CPR checks the existence of such ports in the component domain model, lifting this Product-CPR to a Delta-CPR requires configuration-dependent assignment. The connector addition operation should only be applied for system variants that include both input and output ports. Configuration-dependent assignment places the connector addition operation into a change target delta that is only applied for those system variants.

3.3 Lifting Consistency Preservation Rules

We can lift a Product-CPR to a Delta-CPR and have the lifted Delta-CPR enforce the same CR as the Product-CPR it was lifted from. The Delta-CPR in Listing 2 is created by lifting the Product-CPR from Listing 1. Lifting the Product-CPRs to Delta-CPRs consists of four steps: Replacing the Product-CPR trigger change, replacing the consequential change, lifting the correspondence management, and applying a localization strategy to place the consequential changes into appropriate deltas in the delta domain models. These four steps can all be performed automatically and only require manual implementation effort in cases where external code is called.

Replacing the trigger change. Instead of triggering the Product-CPR when a change to a system variant occurs, a Delta-CPR triggers when a delta operation is added to a delta. The delta operation, whose addition is the trigger change of the Delta-CPR, has to perform the exact same change when applied as the Product-CPR trigger change it replaces. Consider the BCS Product-CPR in Listing 1. The Product-CPR executes after a component is added to a system (Listing 1 line 4). This trigger change gets replaced in the Delta-CPR with the addition of a component addition operation that adds a component to a system (Listing 2 line 5). Additionally, the routine signature is changed to reflect this replacement (Listing 2 line 8).

Replacing the consequential change. A Product-CPR performs a consequential change by applying the consequential change to a domain model, while Delta-CPRs perform a consequential change by adding a delta operation to a delta in a delta domain model. The delta operation added by the Delta-CPR has to perform the exact same change when applied as the consequential change applied to the domain model by the Product-CPR. For the BCS Product-CPR (Listing 1 lines 13-14), the addition of a newly created statemachine region to the statemachine located in the statemachine domain model is replaced by the addition of a delta operation to a delta in the delta statemachine domain model that adds a statemachine region to a statemachine (Listing 2 lines 14-15).

Lifting the correspondence management. Instead of managing correspondences between entities in the variant domain models, Delta-CPRs manage correspondences between delta operations in the delta domain models. A Product-CPR creates a correspondence between two entities in the variant domain models. For the lifted Delta-CPR, this correspondence is replaced by a correspondence between addition delta operations in the delta domain models. For the BCS, the correspondence between a statemachine region and a system component in the Product-CPR (Listing 1 lines 9 and 15) is replaced by a correspondence between the region addition operation and component addition operation in the Delta-CPR (Listing 2 lines 10 and 18).

Applying a localization strategy. The delta operations that get added to a delta domain model as the consequential change of a Delta-CPR must be placed into a delta in a delta domain model of the Var-V-SUM. If the consequential change depends on the system variant it is applied to, configuration-dependent assignment is applied as the localization strategy. Otherwise, delta mirroring or functional delta assignment is applied. The applied localization strategy for the lifted BCS Delta-CPR (Listing 2 lines 16-17) is delta mirroring.

Handling External Code. Product-CPRs might contain calls to external code, such as library calls. When a Product-CPR contains such calls to external code, and the external code is not variability-aware, the external code has to be manually lifted. The Product-CPR in Listing 1 calls the `getContainer()` method in line 10, provided by an external library. This call returns the model entity, in which the model entity provided by the argument is contained. Since this external code is not variability-aware, it needs to be lifted manually to the `getContainerAddOperation()` method (Listing 2 line 11). This lifted method returns the addition operation of the model entity, in which the addition operation provided by the argument adds a new model entity.

3.4 Delta-CPR Correctness

Whenever a change is applied to a delta domain model, it might trigger a Delta-CPR, which applies a consequential change based on the trigger change to a target delta domain model, leading to changes in derivable system variants. If the CR enforced by the Delta-CPR holds for all derivable system variants after the Delta-CPR was executed, the Delta-CPR is *correct*. Showing Delta-CPR correctness for all derivable variants and possible consequential changes is not feasible, since the number of derivable variants can scale exponentially with the number of deltas in a delta domain model, and the number of possible consequential changes can be indefinite. Therefore, we propose to check correctness for a sample of possible system variants for as many consequential changes as we have available. To cover all possible consequential changes of Delta-CPR applications in the delta domain models, the variants in the selected sample have to involve the application of every delta that is present in the delta domain models. We call this selection criteria for system variants in the sample *delta coverage*. The existing system variant domain models before the V-SUM lifting process are combined into system delta domain models after the lifting process. The combination of system variant domain models into a system delta domain model consists of delta operation additions to the delta domain model, which can construct the domain model for all system variants when applied. We propose to use these delta operation

additions as the trigger changes for Delta-CPR applications in order to generate consequential changes for the Delta-CPR correctness check in each domain. Using a sample of system variants with delta coverage and the consequential changes resulting from the additions of all delta operations in each delta domain model results in a high chance of discovering incorrect Delta-CPRs with feasible effort.

4 Implementation

We implement the concepts from Section 3, the Var-V-SUM, lifted Delta-CPRs, and a correctness checking framework of Delta-CPRs [6]. Because the Vitruvius reference implementation already uses the *Eclipse Modeling Framework* (EMF) [10], domain meta-model implementations (including the delta domain meta-models) are instances of the Ecore meta-model, which is provided by EMF.

Variability-aware Virtual Single Underlying Model. By constructing a V-SUM with delta domain models and Delta-CPRs between them, we realize a Var-V-SUM implementation. For the V-SUM, we use the Vitruvius reference implementation [14]. We implement the generic delta meta-model, which includes operations for all possible changes for any generic EMF model based on Delta-Ecore [28]. This generic delta meta-model and any domain meta-model can be used to generate a delta dialect, which includes operations for all possible changes on an instance of that meta-model. By generating delta dialects for both the statemachine and component domain meta-models, we lift all meta-models of the BCS running example to delta domain meta-models.

We lift the required Delta-CPRs for the Var-V-SUM from Product-CPRs. Product-CPRs and Delta-CPRs are both instances of the reaction meta-model provided by Vitruvius. The lifting process is described in Section 3.3. Code that is used multiple times in different Delta-CPRs, such as the localization strategy implementation, is located in external classes to avoid code redundancies in the Delta-CPRs. Cross-domain Delta-CPRs are implemented with delta mirroring or configuration-dependent assignment. All intra-domain Delta-CPRs are implemented with configuration-dependent assignment.

Delta-CPR Correctness Checking Framework. We implement the proposed correctness checking approach (Section 3.4) with the *Delta Consistency Preservation Rule Testing Kit* (Testing Kit) by checking the correctness of a lifted Delta-CPR against the Product-CPR it was lifted from. If this Product-CPR preserves exactly the same CR on variants as the lifted Delta-CPR, they have two properties resulting from the lifting process described in Section 3.3. The trigger change for the Product-CPR is the application of the delta operation, whose addition is the trigger change for the Delta-CPR. And the consequential changes for the Product-CPR are the applications of delta operations, whose addition is the consequential change of the Delta-CPR. The Testing Kit uses these two properties to check for the correctness of lifted Delta-CPRs. If the delta operations, added by consequential changes of a Delta-CPR, are placed in a previously empty target delta domain model, variant domain models derived from these target delta domain models represent the applied delta operations added by consequential changes, which modify a specific system variant. By applying the consequential changes of Product-CPRs to previously empty target delta domain models, this approach allows us to compare the delta domain models per variant, which exclusively include applied consequential changes of Delta-CPRs and Product-CPRs. Adding the deltas to empty delta domain models to trigger Delta-CPRs, or applying the deltas to empty domain models to trigger Product-CPRs, ensures the target domain models are empty because no previous changes resulted in consequential changes to the target domain models.

The Testing Kit consists of V-SUMs using the Vitruvius reference implementation [14], a Var-V-SUM implementation, and a system variant comparator. For the evaluation in Section 5.3, we use a BCS case study specific variant comparator that identifies model entities based on their name, except for entities without unique names. For example, the BCS case study [18] sets connector names based on the signal they carry. The comparator implementation does not compare these connectors based on their name but based on the combination of source port, target port, and signal. The comparator itself is an extension of the `EcoreUtil.EqualityHelper`, which is a standard comparison tool for EMF models. This comparator is the only part of the Testing Kit that needs to be adjusted when applying the Testing Kit to other case studies.

When applying the Testing Kit for checking the S2C Delta-CPR correctness for system variants P_0 - P_{17}, which are provided by the BCS case study [18] and achieve delta coverage, the Testing Kit uses the addition of every delta operation in the BCS case study statemachine deltas to the statemachine delta domain model as the trigger changes for the Delta-CPR applications and the applications of those delta operations (the stepwise construction of the statemachine domain models) as the trigger changes for the Product-CPR applications. The Product-CPRs react to the construction of the statemachine domain models for each system variant and apply the consequential changes to the component domain models for that system variant (top). The Delta-CPRs react to the addition of all delta operations to the statemachine delta domain model and apply the consequential changes to the component delta domain model (bottom). After the Product-CPRs and Delta-CPRs applied consequential changes, we derive system variants P_0 - P_{17} from the Var-V-SUM and compare their component domain models to the component domain models of system variants P_0 - P_{17} contained in the V-SUMs. If the comparison shows the component domain models are identical, then we know the CR preserved by the Product-CPR was also preserved by the Delta-CPR on system variants, hence showing the correctness of the Delta-CPR.

5 Evaluation

We evaluate the correctness for the lifted Delta-CPRs contained in our case study and examine scalability by evaluating the number of consequential changes performed by Delta-CPRs compared to the number of consequential changes performed by Product-CPRs. These evaluation criteria were selected to show the feasibility of using a Var-V-SUM to develop variant-rich systems.

5.1 Research Questions

To evaluate if Delta-CPRs work as expected and can be practically used, we aim to answer the following two research questions:

- **RQ1 Soundness: Can we lift Product-CPRs to Delta-CPRs and achieve correctness?** This is the central research question for our evaluation. The Delta-CPR application has to lead to consistent system variants. Otherwise, a single Var-V-SUM is not a sufficient replacement for one V-SUM per system variant. To answer this research question, we compare consequential changes performed by lifted Delta-CPRs to the consequential changes performed by the Product-CPRs, from which they were lifted.
- **RQ2 Scalability: Is the application of Delta-CPRs feasible for large systems?** Delta-CPRs handle correspondences and configuration-dependent changes differently compared to Product-CPRs. Their consequential changes, including the correspondence model, are of different size and complexity. If scalability in these areas is not given, the Var-V-SUM approach is not applicable to modeling projects of large sizes. To answer this research question, we compute the number of entities in domain and correspondence models created by lifted Delta-CPRs and compare them to the number of entities in domain and correspondence models created by Product-CPRs, from which they were lifted.

5.2 Subject System

We use a product representation and a delta representation of the BCS to answer our research questions. The BCS product representation includes 18 of the 11616 system variants ($P_0 - P_{17}$ presented in the BCS report [18]), each with a statemachine domain model, component domain model, and Product-CPRs. These 18 system variants are used as samples for correctness checking of Delta-CPRs since they fulfill the proposed delta coverage selection criteria. The 18 sample system variants also fulfill pairwise feature interactions coverage, which have been shown to achieve good results for testing variable systems [22]. For the statemachine and component domain, we implemented 20 Product-CPRs, listed in Table 1. Eleven of these Product-CPRs synchronize the existence of entities in domain models, six synchronize the entity attributes in domain models, and three are intra-domain reactions in the component domain model. All 20 Product-CPRs create and retrieve correspondences from the correspondence model. The delta representation includes the statemachine delta domain model and the component delta domain model. Deltas included in these domain models can be used to derive all 11616 valid variants of the BCS, including variants P_0 to P_{17}.

5.3 Experiment Setup and Procedure

We conduct a single experiment to answer both research questions RQ1 and RQ2 and perform the following steps:

(1) We lift the 20 Product-CPRs to 20 Delta-CPRs by applying the steps described in Section 3.3. We check the correctness of these lifted Delta-CPRs in steps (2)-(4) by applying the Testing Kit described in Section 4. When a CR does not hold for a domain model, the *consistency-preserving change* is the change that reestablishes the CR. The Delta-CPR correctness check requires comparing consistency-preserving changes generated by the lifted Delta-CPRs to those generated by the Product-CPRs, for each BCS variant P_0 to P_{17}. Directly comparing consistency-preserving changes is difficult for this evaluation, so instead we apply them to variant domain models and compare these domain models.

(2) We generate and apply consequential changes of Product-CPRs in V-SUMs to have a baseline for consistency-preserving changes. We apply the deltas necessary to build the source domain models of a variant and apply the consequential changes to empty target domain models. For each variant, this results in the application of consequential changes to one target statemachine domain model and one target component domain model. For the system variants P_0 to P_{17}, this results in the consequential change application to 18 target statemachine domain models and 18 target component domain models. These domain models represent the applied consistency-preserving changes for each variant, generated by the Product-CPRs in V-SUMs.

(3) We generate and apply consequential changes of Delta-CPRs in a Var-V-SUM to derive consistency-preserving changes, which we can compare against the baseline. We add all BCS deltas to the component and statemachine source delta domain models and apply the consequential changes to the empty statemachine and component target delta domain models. We derive statemachine domain models and component domain models for BCS variants P_0 to P_{17} from the

Table 1: List of all implemented Product-CPRs for the BCS applied to the statemachine domain models and component domain models. They are divided into three types of Product-CPRs: (E) for synchronization of entity existence, (A) for synchronization of entity attributes, and (I) for intra-domain consistency.

	Trigger Change	Consequential Change
E	Add Statemachine	Add System
E	Add Com-Signal	Add Signal
E	Add Region	Add Component
E	Add Transition Trigger Signal	Add Input Port
E	Add Transition Behavior Signal	Add Output Port
E	Remove Transition	Remove In/Output Port
E	Add System	Add Statemachine
E	Add Signal	Add Com-Signal
E	Remove Signal	Remove Com-Signal
E	Add Component	Add Region
E	Remove Component	Remove Region
A	Change Statemachine name	Align System name
A	Change Com-Signal name	Align Signal name
A	Change Region name	Align Component name
A	Change System name	Align Statemachine name
A	Change Signal name	Align Com-Signal name
A	Change Component name	Align Region name
I	Set Port Signal	Add Connector
I	Remove Input Port	Remove Connector
I	Remove Output Port	Remove Connector

target statemachine delta domain model and target component delta domain model. These derived domain models represent the applied consistency-preserving changes for each variant, generated by the Delta-CPRs.

(4) After the derivation of domain models, we compare the domain models containing the applied consistency-preserving changes generated by the Product-CPRs of step (2) to the derived domain models containing the applied consistency-preserving changes generated by the Delta-CPRs of step (3) for each of the BCS variants P_0 to P_{17}. Comparing the target statemachine domain model with the derived statemachine domain model and the target component domain model with the derived component domain model for each of the BCS variants P_0 to P_{17} results in a total of 36 domain model comparisons.

(5) We conclude the experiment by computing the number of entities and correspondences that were created by Product-CPR applications in the target domain models and Delta-CPR applications in the target delta domain models.

5.4 Results

RQ1 Soundness: If the domain models consisting of consistency-preserving changes for a BCS variant generated by the Product-CPRs and generated by the Delta-CPRs are identical, they preserve the same CRs. If the Product-CPRs and the lifted Delta-CPRs preserve the same CRs for all BCS variants, the Delta-CPRs are correct, and the lifting process was successful. The domain models are identical for all 36 domain model comparisons between the target domain models and the domain models derived from target delta domain models. The 36 derived domain models achieve delta coverage, so every delta with consequential changes in the target domain delta model of Delta-CPRs was applied to derive models for the 36 domain model comparisons. Because of the comparison results, we can conclude the Product-CPRs and the lifted Delta-CPRs preserve the same CRs. For the lifting process from Product-CPRs to Delta-CPRs, two external EMF utility methods had to be manually lifted, with the external methods comprising less than 100 lines of Java code combined. The remaining liftings can be performed automatically.

RQ2 Scalability: The consequential changes include the addition of domain model entities for Product-CPRs, the addition of delta domain entities for Delta-CPRs, and the addition of correspondences into the correspondence model for both. Evaluating the scalability of Delta-CPRs requires comparing the number of entities and correspondences created by Delta-CPRs to the number of entities and correspondences created by Product-CPRs added up over all BCS variants because the delta domain models can be used to derive domain models for all variants. Since the computation for all BCS variants is unfeasible, we only compute the number of entities and correspondences created by Product-CPRs for the BCS variants P_0 to P_{17} and apply ordinary least squares linear regression to approximate the number of created entities and correspondences for all possible BCS variants, if necessary. Table 2 shows the computed sums for the entities in domain models and delta domain models and correspondences in the correspondence models, only considering BCS variants P_0 to P_{17}. The Product-CPRs create more correspondences than the Delta-CPRs but fewer domain model entities, than the Delta-CPRs delta domain model entities for BCS variants P_0 to P_{17} in our experiment. Figure 4 plots model entities and correspondences, summed up for a subset of system variants (P_0 up to P_x), with the greatest plotted subset being P_0 to P_{17}. For example, the subset of domain models with six variants in the left plot of Figure 4 is the sum of entities in the models, containing the consistency-preserving changes for BCS variants P_0 up to P_5. Since the delta domain models can be used to derive any possible system variant, the values for Delta-CPRs are constant in these plots independent of the system variant subset. The plot shows seven variants are necessary until the Product-CPRs create more correspondences than Delta-CPRs in our case study. The number of variants necessary, until the Product-CPRs created more domain model entities than Delta-CPRs in our case study, is greater than 18 and not reached in our experiment. We apply ordinary least squares linear regression to the data points of BCS variants P_0 to P_{17} to estimate the break-even point for the number of domain model entities to be around 37 of the 11616 possible variants.

Table 2: Sum of domain model entities and correspondences in the 36 models containing the consistency-preserving changes of the Product-CPRs compared to delta domain model entities and correspondences in the two models containing the consistency-preserving changes of the Delta-CPRs.

	∑#Entities	∑#Correspondences
Product-CPRs	4527	7376
Delta-CPRs	9529	2450

5.5 Discussion

RQ1 Soundness: Can we lift Product-CPRs to Delta-CPRs and achieve correctness? We answer this research question positively. Product-CPRs can be lifted to correct Delta-CPRs that preserve the same CR on system variants as the Product-CPRs, which they originate from. The correctness of lifted Delta-CPRs can be checked by applying the Testing Kit presented in Section 4. We conclude the correctness of the Delta-CPRs in our evaluation because all Product-CPRs and Delta-CPRs were executed at least twice when applying the Testing Kit for BCS variants P_0 to P_{17}. Additionally, the lifting can be performed with low effort. The only manual implementation effort required to lift all 20 Product-CPRs comprises less than 100 lines of Java code. 14 of the 20 Product-CPRs were lifted by applying delta mirroring as the localization strategy. The remaining six Product-CPRs required the usage of configuration-dependent assignment as the localization strategy.

RQ2 Scalability: Is the application of Delta-CPRs feasible for large systems? We also answer this research question positively. The Delta-CPR application scales well for large variant-rich systems, which makes a Var-V-SUM feasible. The reuse aspect of delta-oriented variability modeling applies to the number of entities added by consequential changes of Delta-CPRs, since a single delta operation added to a delta domain model can be applied to

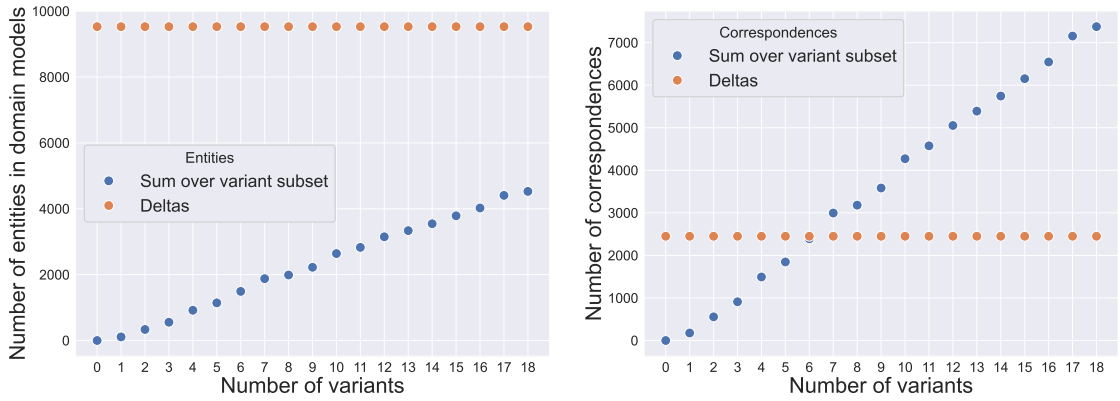


Figure 4: Comparison for the number of domain model entities summed up for a subset of system variants (P_0 up to P_x) on the left side, and the number of correspondences in the correspondence model summed up for a subset of system variants on the right side, to the number of domain model entities and correspondences for using lifted delta models.

derive multiple different system variants. A component addition delta operation is added to a delta domain model by a Delta-CPR a single time, while the component itself can be added to domain models of multiple variants by a Product-CPR. This reuse aspect does not apply to consequential changes of Delta-CPRs that are required to use configuration-dependent assignment, since in the worst case these consequential changes depend on a single variant and therefore get added once per system variant. The application of configuration-dependent assignment for Delta-CPRs leads to a number of entity additions to target delta domain models so high, we were unable to reach the break-even point with the BCS case study variants P_0 up to P_{17}. But with a break-even point of about 37 variants and 11616 valid possible variants of the BCS, the application of Delta-CPRs scales overall well in the number of entity additions to the target delta domain models. Additionally, the number of deltas created with configuration-dependent assignment could be reduced by refactoring the delta domain models [27].

The reuse aspect of delta-oriented variability modeling also applies to the number of correspondences in the correspondence model that get added by Delta-CPRs. The lifting of correspondence management discussed in Section 3.3 leads to the addition of correspondences for addition operations, instead of correspondences for the entities these operations add. Adding a component to component domain models leads to the addition of a correspondence in the correspondence model for each system variant that includes that component (see Listing 1). In contrast, adding a component addition delta operation leads to the addition of a single correspondence in the correspondence model (see Listing 2). The reuse of correspondences between variants leads to reaching the break-even point for just seven variants in our BCS case study.

The reuse of model entities and correspondences directly translates to a lower number of model changes necessary to apply consequential changes. A reduced number of model changes influences the runtime and memory requirements for consistency preservation, since the Vitruvius consistency preservation mechanism operates linearly in the number of model changes [12] on models in memory.

5.6 Threats to Validity

Internal Validity. The correctness of the Delta-CPR correctness checking implementation (see Section 4), specifically the model comparator implementation, is a threat to internal validity for RQ1. With a faulty comparator implementation, the check for Delta-CPR correctness would not be conclusive, and RQ1 could not be answered. The comparator implementation was tested multiple times during the development, including automated [6] and manual tests on different system variants of the BCS case study. It was implemented by following software engineering best practices and extends a standard EMF model comparison tool to reduce error probability in the core functionality.

The granularity of the evaluation process is another threat to internal validity for RQ1. Instead of checking the correctness for a single Delta-CPR at a time, we check the correctness for multiple Delta-CPRs that add the delta operations used to derive a domain model for the comparison. All Product-CPRs, and therefore lifted Delta-CPRs, have different consequential changes (see Table 1). We implement additional automated tests [6] that ensure each Product-CPR and lifted Delta-CPR only perform the exact consequential changes listed in Table 1.

The metric selection of model entities and correspondences is a threat to internal validity for RQ2. If the chosen metrics do not translate to scalability in practice, collecting them is not helpful to determine the scalability of our approach. The number of entities in a model is a meaningful metric because it gives a sufficient estimation of model size and complexity. The number of correspondences is another meaningful metric, because *Vitruvius* has to maintain the correspondence model in addition to the domain models. If the number of correspondences increases, the preservation of consistency becomes more extensive.

External Validity. The representativeness of the BCS case study for multi-domain variant-rich systems is a threat to external validity for RQ2. The size of the BCS is limited and has a clear one-to-one mapping between different domain model elements. The findings

might not be fully applicable to large domain models with one-to-many mappings between domain model elements. While we did show feasibility of this approach for large systems, further application to more complex case studies is required.

6 Related Work

Multi-domain consistency preservation is a long-standing problem but has not been thoroughly adapted to variant-rich systems.

Preserving Multi-Domain Consistency. Systematic literature research into multi-domain consistency preservation approaches [29] shows strengths, limitations, and tool support for different approaches to the consistency preservation problem. While multiple existing approaches could be adapted to be applicable to variant-rich systems, we choose *Vitruvius* [15] for our adaptation because Vitruvius represents changes similarly to deltas in delta-oriented variability modeling, and Vitruvius provides an open-source reference implementation [14] that is extendable with open-source EMF tooling [10]. Vitruvius combines meta-models of different domains into a virtual single underlying model with consistency preservation rules, making them appear from the outside as a single underlying model [4] without redundancies.

Extending the Delta Modeling Approach. Operations on deltas in the form of addition, removals, and changes are explored in [17]. These operations on deltas represent delta operations on a delta model, called *higher-order delta operations*. A Delta-CPR is executed when a higher-order delta operation is applied and reacts with the application of higher-order delta operations as the consequential changes.

Combining Multi-Domain Consistency with Variability. While there are many approaches to check for consistency in variable systems [24], these do not address consistency preservation between domain models. Lopez-Herrejon and Egyed [19] [20] consider multi-domain consistency checking for variant-rich systems and discuss possible localization strategies for multi-domain consistency preservation for feature-oriented programming [5]. However, these works do not propose any multi-domain consistency preservation mechanisms for variant-rich systems. Kowal and Schaefer [16] categorize CRs for variant-rich multi-domain systems into intra-domain, cross-domain, and cross-variant CRs. Kowal and Schaefer do not address the preservation of those CRs, instead focusing on checking if the CRs apply. Of these three categories, the Var-V-SUM approach is able to preserve intra-domain and cross-domain CRs.

Ananieva et al. propose *VaVe* [2], an approach for the management of system variants and system versions in a V-SUM. VaVe does not solve the research problem of multi-domain consistency preservation on variant-rich systems because it only introduces new consistency checks between variants and versions of a single domain while keeping the Vitruvius multi-domain consistency preservation mechanism untouched. This makes the approach not applicable for multi-domain consistency preservation in variant-rich systems. Neumann et al. [21] present an approach that uses delta-oriented variability modeling and Vitruvius, but for a different purpose. Neumann et al. translate consequential changes of Product-CPRs on system variant domain models to deltas. These deltas represent the consistency-preserving changes per system variant and can be applied to obtain consistent system variants. Our approach aims to apply consequential changes on delta domain models, which leads to derivations of already consistent system variants from deltas. A preliminary version of our approach has been presented in [7].

7 Conclusion

The Var-V-SUM represents the combination of the delta-oriented variability modeling [17] [9] and Vitruvius [15] multi-domain consistency preservation approaches. Introducing a Var-V-SUM enables consistency preservation for variant-rich multi-domain systems by using delta domain models and Delta-CPRs. Delta-oriented variability modeling can be applied to lift domain models to delta domain models and represent a domain for the complete variant-rich system instead of just a single system variant. Product-CPRs preserve CRs for a single system variant by applying changes to the domain models of that variant. Such a Product-CPR, which is only applicable to a single system variant, can always be lifted to a Delta-CPR. Delta-CPRs preserve CRs for every system variant simultaneously by applying changes to the delta domain models of the system. The lifting process of domain models to delta domain models is automatically performed by DeltaEcore [28]. The lifting process of Product-CPRs to Delta-CPRs requires performing lifting steps described in Section 3.3, which can be performed with small effort or automatically if no external method calls are involved in the Product-CPR applications.

By using a Var-V-SUM, we combine the advantages of delta-oriented variability modeling and Vitruvius. The reuse of core model entities shared between system variants enables the development of variant-rich systems without redundant modeling, while Vitruvius enables consistency preservation for multi-domain models. A Var-V-SUM enables consistency preservation for variant-rich multi-domain systems without requiring redundant modeling and redundant Product-CPR applications.

The Var-V-SUM as presented in this work should be understood as a highly expandable approach. Deltas can describe variability in space (*variants*), but also variability in time (*versions*) [1]. Since Delta-CPRs do not make assumptions about what a delta represents, the Var-V-SUM approach is applicable to preserve consistency for variants and versions. Vitruvius allows the definition of views. Product-oriented development and platform-oriented development [3] of *Software Product Lines* (SPLs) are both achievable with a Var-V-SUM. Delta-CPRs can be used to preserve consistency between domain models in the solution space of an SPL, while variability in the problem space can be modeled with additional variability models, such as *feature models*, in the Var-V-SUM. Consistency preservation between the problem and solution space of a SPL could be realized with the introduction of additional CPRs between models in the problem space and delta domain models in the solution space of the SPL, as described in Eger et al. [11].

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1608 – 501798263 and the KASTEL Security Research Labs.

References

[1] Sofia Ananieva, Sandra Greiner, Thomas Kühn, Jacob Krüger, Lukas Linsbauer, Sten Grüner, Timo Kehrer, Heiko Klare, Anne Koziolek, Henrik Lönn, Sebastian Krieter, Christoph Seidl, S. Ramesh, Ralf Reussner, and Bernhard Westfechtel. 2020. A conceptual model for unifying variability in space and time. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*. ACM, Montreal Quebec Canada, 1–12. doi:10.1145/3382025.3414955

[2] Sofia Ananieva, Heiko Klare, Erik Burger, and Ralf Reussner. 2018. Variants and Versions Management for Models with Integrated Consistency Preservation. In *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems*. ACM, Madrid Spain, 3–10. doi:10.1145/3168365.3168377

[3] Michał Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Ștefan Stănciulescu, Andrzej Wąsowski, and Ina Schaefer. 2014. Flexible product line engineering with a virtual platform. In *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, Hyderabad India, 532–535. doi:10.1145/2591062.2591126

[4] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. 2010. Orthographic Software Modeling: A Practical Approach to View-Based Development. In *Evaluation of Novel Approaches to Software Engineering*, Leszek A. Maciaszek, César González-Pérez, and Stefan Jablonski (Eds.). Vol. 69. Springer Berlin Heidelberg, Berlin, Heidelberg, 206–219. Series Title: Communications in Computer and Information Science. doi:10.1007/978-3-642-14819-4_15

[5] D. Batory, J.N. Sarvela, and A. Rauschmayer. 2004. Scaling step-wise refinement. *IEEE Transactions on Software Engineering* 30, 6 (June 2004), 355–371. doi:10.1109/TSE.2004.23

[6] Simon Bothe and Dirk Neumann. 2026. Replication Package for "Lifting Multi-Domain Consistency Preservation to Variant-Rich Systems". doi:10.5281/ZENODO.21064493

[7] Bothe, Simon. 2026. Enabling Cross-Domain Consistency Preservation for Delta-Oriented Product Line Development. Master thesis, Karlsruhe Institute of Technology. doi:10.5445/IR/1000195066

[8] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice*. Springer International Publishing, Cham. doi:10.1007/978-3-031-02549-5

[9] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. 2015. Abstract delta modelling. *Mathematical Structures in Computer Science* 25, 3 (March 2015), 482–527. doi:10.1017/S0960129512000941

[10] Eclipse Foundation. 2025. Eclipse Modeling Framework: 2.43. https://eclipse.dev/emf

[11] Fabian Eger, Lars König, Thomas Weber, Dirk Neumann, Erik Burger, Kevin Feichtinger, and Ralf Reussner. 2026. Towards Consistent, Variability-Aware Views in Systems Development. Limassol Cyprus. accepted for publication at the International Conference on Software and Systems Reuse, Product Lines, and Configuration (VARIABILITY 2026).

[12] Benedikt Jutz and Thomas Weber. 2024. Scalability of Consistency Preservation with Vitruvius. In *Softwaretechnik-Trends Band 44, Heft 4*. Gesellschaft für Informatik e.V.

[13] Cory Kapser and Michael Godfrey. 2006. "Cloning Considered Harmful" Considered Harmful. In *2006 13th Working Conference on Reverse Engineering*. IEEE, Benevento, Italy, 19–28. doi:10.1109/WCRE.2006.1

[14] Heiko Klare, Joshua Gleitze, Max Kramer, Lukas H, Dominik Werle, Timur Sağlam, Rolleander, Lars König, Arne Lange, Thomas Weber, Stephan Seifermann, AndreasLoeffler, thomyxx, Martin Armbruster, Jan Wittler, Leonie Schmidt, NeumannDirk, Benedikt Jutz, Dominik, Muhammad-Tahir-KIT, Nathan Hagel, Dominik Klooz, Fabian Scheytt, SofiaAnanieva, h4uges, and tools vitruv. 2025. vitruv-tools/Vitruv: 3.2.2. doi:10.5281/ZENODO.15855904

[15] Heiko Klare, Max E. Kramer, Michael Langhammer, Dominik Werle, Erik Burger, and Ralf Reussner. 2021. Enabling consistency in view-based system development — The Vitruvius approach. *Journal of Systems and Software* 171 (Jan. 2021), 110815. doi:10.1016/j.jss.2020.110815

[16] Matthias Kowal and Ina Schaefer. 2016. Incremental Consistency Checking in Delta-oriented UML-Models for Automation Systems. *Electronic Proceedings in Theoretical Computer Science* 206 (March 2016), 32–45. arXiv:1604.00348 [cs]. doi:10.4204/EPTCS.206.4

[17] Sascha Lity, Matthias Kowal, and Ina Schaefer. 2016. Higher-order delta modeling for software product line evolution. In *Proceedings of the 7th International Workshop on Feature-Oriented Software Development*. ACM, Amsterdam Netherlands, 39–48. doi:10.1145/3001867.3001872

[18] Sascha Lity, Remo Lachmann, Malte Lochau, and Ina Schaefer. 2017. Delta-oriented software product line test models-the body comfort system case study. https://github.com/TUBS-ISF/BCS-Case-Study-Full

[19] Roberto Erick Lopez-Herrejon and Alexander Egyed. 2010. Detecting Inconsistencies in Multi-View Models with Variability. In *Modelling Foundations and Applications*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Thomas Kühne, Bran Selic, Marie-Pierre Gervais, and François Terrier (Eds.). Vol. 6138. Springer Berlin Heidelberg, Berlin, Heidelberg, 217–232. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-642-13595-8_18

[20] Roberto E. Lopez-Herrejon and Alexander Egyed. 2012. Towards fixing inconsistencies in models with variability. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*. ACM, Leipzig Germany, 93–100. doi:10.1145/2110147.2110158

[21] Dirk Neumann, Tobias Pett, and Ina Schaefer. 2025. Towards Consistency Preservation for Multi-Domain Variability Modeling. In *Proceedings of the 29th ACM International Systems and Software Product Line Conference - Volume A*. ACM, A Coruña Spain, 88–93. doi:10.1145/3744915.3748466

[22] Sebastian Oster, Marius Zink, Malte Lochau, and Mark Grechanik. 2011. Pairwise feature-interaction testing for SPLs: potentials and limitations. In *Proceedings of the 15th International Software Product Line Conference, Volume 2*. ACM, Munich Germany, 1–8. doi:10.1145/2019136.2019143

[23] Alexander Pretschner, Manfred Broy, Ingolf H. Kruger, and Thomas Stauner. 2007. Software Engineering for Automotive Systems: A Roadmap. In *Future of Software Engineering (FOSE '07)*. IEEE, Minneapolis, MN, USA, 55–71. doi:10.1109/FOSE.2007.22

[24] Alcemir Rodrigues Santos, Raphael Pereira De Oliveira, and Eduardo Santana De Almeida. 2015. Strategies for consistency checking on software product lines: a mapping study. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*. ACM, Nanjing China, 1–14. doi:10.1145/2745802.2745806

[25] Ina Schaefer, Lorenzo Bettini, Viviana Bono, Ferruccio Damiani, and Nico Tanzarella. 2010. Delta-Oriented Programming of Software Product Lines. In *Software Product Lines: Going Beyond*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Jan Bosch, and Jaejoon Lee (Eds.). Vol. 6287. Springer Berlin Heidelberg, Berlin, Heidelberg, 77–91. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-642-15579-6_6

[26] Ina Schaefer and Ferruccio Damiani. 2010. Pure delta-oriented programming. In *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development*. ACM, Eindhoven The Netherlands, 49–56. doi:10.1145/1868688.1868696

[27] Sandro Schulze, Oliver Richers, and Ina Schaefer. 2013. Refactoring delta-oriented software product lines. In *Proceedings of the 12th annual international conference on Aspect-oriented software development*. ACM, Fukuoka Japan, 73–84. doi:10.1145/2451436.2451446

[28] Christoph Seidl, Ina Schaefer, and Uwe Aßmann. 2014. DeltaEcore-a model-based delta language generation framework. In *Modellierung 2014*. Gesellschaft für Informatik e.V., Bonn, 81–96. tex.pissn: 1617-5468.

[29] Weslley Torres, Mark G. J. Van Den Brand, and Alexander Serebrenik. 2021. A systematic literature review of cross-domain model consistency checking by model management tools. *Software and Systems Modeling* 20, 3 (June 2021), 897–916. doi:10.1007/s10270-020-00834-1