Evaluating Consistency Preservation for Multi-Domain Variability Propagation

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

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

Today, model-driven engineering of variant-rich systems involves parallel development in multiple domains such as electronics, mechanics, and software. Rapid evolution can lead to inconsistencies, such as conflicting system models, which can have disastrous consequences in safety-critical functions (e.g. vehicle brakes). Even though partial solutions to both: single-domain variability modeling, and multi-domain consistency preservation exist, their combination remains unexplored. This paper presents a combined approach to *consistency-preserving multi-domain* ***variability propagation*** for variant-rich multi-domain systems. We exploit the incremental nature of two existing approaches. First, *delta modeling*, is a variability modeling technique to derive one product variant from another through the application of changes in the form of *deltas*. Second, the *Vitruvius* approach, which preserves multi-domain consistency via *consistency preservation rules (CPRs)* that react to model changes. The combination of both enables applying deltas in a source domain, which, via CPRs, leads to applying deltas in the target domain. We evaluate soundness and completeness of variability propagation as well as the influence of CPRs on both former properties. Soundness is assessed by checking whether propagated deltas and product variants are correct subsets of the respective manually written ones. Completeness measures how many expected elements of a delta/ product variant can be derived through variability propagation. Using an industry-related case study that models the variant-rich functionality of a vehicle's body-comfort-system in multiple domain, results show that variability propagation yields sound target-domain deltas and product variants independent of CPRs. However, completeness is limited, and CPRs strongly influence the amount of required manual completion.

CCS Concepts

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



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/10.1145/3837062.3839065

Keywords

Multi-Domain Modeling, Incremental Consistency Preservation, Variability Propagation, Delta Modeling

ACM Reference Format:
Dirk Neumann, Simon Bothe, Tobias Pett, and Ina Schaefer. 2026. Evaluating Consistency Preservation for Multi-Domain Variability Propagation. 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/10.1145/3837062.3839065

1 Introduction

Consistency is essential for the development of multi-domain systems, such as cyber-physical systems (CPS). For example, a car is described using multiple models, each addressing a different domain like electronics, physics, or informatics [6, 24]. The domain models must be kept consistent to avoid failures during construction or operation. While this is already a challenging task, it becomes even more difficult if not only a single car, but a variant-rich system of multiple car variants are simultaneously managed, e.g., in a product line [23]. Parallel development in multiple domains of variant-rich systems can lead to conflicting models of different domains. For example, a software engineer could modify the behavior of an existing sensor, while an electrical engineer swaps the old sensor for a new one, effectively making software and hardware incompatible. Conveniently, consistency relations between multiple domains are often well understood and sometimes even explicitly formulated. This knowledge can be operationalized to actively preserve consistency instead of only checking for it. However, existing approaches to multi-domain consistency preservation are often not variability-aware. Conversely, traditional approaches to (software) product line engineering are only concerned with a single modeling-domain, mostly software. Failing to manage both facets in parallel increases the risk of inconsistencies, potentially leading to high costs and disastrous consequences.

However, reusing existing approaches can combine their advantages while mitigating their shortcomings. Previously, a novel approach to *consistency-preserving multi-domain variability modeling* [20] was proposed, which aims to improve the development of variant-rich multi-domain systems. This approach incorporates the existing technique of *incremental consistency preservation* [8] into the well-known variability modeling technique *delta modeling*

[25, 26]. Both existing approaches can easily be combined, since they both work based on incremental changes applied to a model. For incremental consistency preservation, different types of changes in the source domain are defined as triggers. If they are activated, consistency repair routines are executed, which apply changes in a target domain to reestablish consistency. In delta modeling, product variants are derived by incrementally applying deltas, which encapsulated multiple changes, to an existing product variant. By triggering consistency repair routines during the product derivation with deltas, changes are applied in the target domain to preserve consistency. Capturing these changes and persisting them as deltas allows to re-apply them in other product derivation steps. Thus, the variability of the source domain delta effectively gets propagated to the target domain. This will not only reduce modeling effort, but also guarantee consistency, effectively improving the development in two different ways.

This paper aims to implement and evaluate the proposed approach [20], which henceforth we will call *variability propagation*. We utilize the *Vitruvius approach* [13] and its reference implementation `vitruv-tools`[1] for incremental consistency preservation and combine it with our implementation[2] of *delta modeling*. Using this implementation, we propagate variability in the form of deltas from one domain to another. For the evaluation, we use the *Body Comfort System* (BCS) as an exemplary proof-of-concept case study. The BCS was developed in cooperation with the automotive industry and models variant-rich usability functionality of a vehicle. It consists of two domains which are abstractions for structural component based architectures and behavior descriptions. We assess propagated deltas regarding soundness and completeness. Soundness is assessed by comparing synthesized deltas and product variants derived with them. If they contain only delta operations or model elements respectively which are present in the manually modeled ground truth, they are sound. Similarly, propagated variability is considered complete, if a synthesized delta and the product variant derived from it contain all delta operations or model elements respectively which are present in the modeled ground truth. In addition, we investigate the influence of the consistency definitions between the modeling domains involved on the assessed properties. Since the consistency definition is both: the description of a desired state and the main instruction artifact of how to reach this state, it is likely to have a big influence on the mapped out approach.

In summary, we make the following contributions:

- We provide fundamental terminology on modeling consistency and variability modeling (Section 2).
- We introduce the existing approach to variability propagation and illustrate it with a running example from the BCS (Section 3).
- We present our implementation of delta modeling and combine it with `vitruv-tools` to provide an implementation of variability propagation (Section 4).
- We assess the soundness and completeness of propagated deltas compared to manually modeled deltas and determine the influence of consistency preservation on these properties (Section 5).

[1] https://github.com/vitruv-tools
[2] https://doi.org/10.5281/zenodo.21295984

2 Background

Multi-Domain Systems and Consistency. In model-driven engineering, *models* are the primary domain artifacts used to describe a system and to derive other artifacts, such as source code. Each model conforms to a corresponding *meta-model*, which defines its structure. A multi-domain system consists of multiple models which may conform to different meta-models. This becomes especially complex when different engineering disciplines like software, electrical, or mechanical engineering use domain-specific modeling languages and standards rather than a common standard, as for example UML. In such a multi-domain system, consistency is a key property. Within a single domain, models must correctly represent their respective system and conform to their meta-model. Across multiple domains, different models may represent the same information of a system. Their meta-models may address different concerns in a systems description, but still have some overlap between them. Here, consistency preservation becomes essential. Changing one model may require the adapting of another model so that information in them do not conflict. This issue has been addressed using different approaches. *Orthographic Software Modeling* (OSM) [5] solved this problem by avoiding overlaps between different domains' meta-models. The introduced *Single Underlying (Meta-)Model* SU(M)M is defined to be free of redundancy and therefore inconsistencies. The development on a SUM is realized via projections and committing changes on them ensures that consistency is always preserved. However, OSM assumes a redundancy-free SUMM which does not represent reality where domain-specific meta-models are used, e.g., because of domain-specific standards or to only disclose parts of used data structures to third parties. The *Vitruvius* approach [13] relaxed the assumptions of the SUMM and introduces the *Virtual Single Underlying (Meta-)Model* V-SU(M)M. Here, existing meta-models can be reused by combining them to a loosely coupled V-SUMM. So-called *Consistency Preservation Rules* (CPR) are used to address the now potentially arising inconsistencies due to redundancies. CPRs define *reactions* which react to potentially consistency-breaking *original changes* in a source domain. These reactions then call *routines* which execute *consequential changes* in the target domain. Between each pair of source and target domain one set of CPRs can be defined.

Delta-Oriented Variability Modeling. Following the Vitruvius approach, a single product, represented by a V-SUM, can be maintained. However, multiple *product variants* may be maintained in parallel for various reasons. User customizability, deployment under different local policies or technical interoperability (e.g., different operating systems) introduces variability. Effective management of the resulting variant-rich system, such as a *product line* is needed. Software Product Line Engineering [23] is a discipline that aims to effectively manage variability. It addresses the *solution space* consisting of the *implementation* of variable artifacts and the *product derivation* with variable artifacts. One approach to realize this for software is *delta-oriented programming* [26]. Here, (atomic) changes, called *delta operations*, are grouped into *deltas* which represent a user intent and can then be reused to derive different product variants by incrementally applying the delta operations to a piece of software program. As stated previously, software can be seen as another model of a system, and so delta-oriented programming

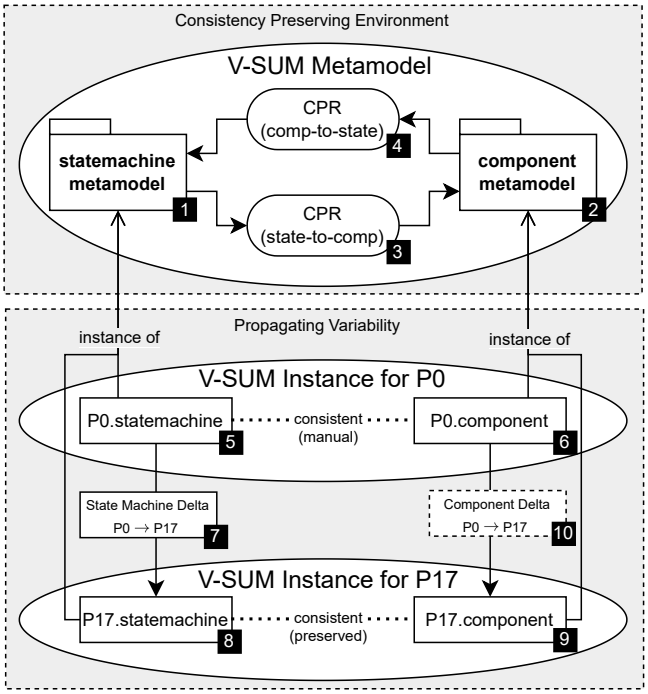


Figure 1: Consistency-preserving variability propagation across domains using the Vitruvius approach [13].

can be generalized to *delta-oriented variability modeling* [25]. Here, incremental delta operations/ changes are applied to one *model variant* to derive another model variant. The incremental application of changes in this variability modeling approach integrates extremely well with the consistency preservation mechanism of the Vitruvius approach.

3 Concept

We aim to evaluate the approach which was proposed in [20]. Therefore, we give a short overview over it here, using the same running example, the BCS. The BCS consists of models of two different domains: the *component domain* which describes the components of a system as well as their interface specifications and the *statemachine domain* which describes their corresponding behavior. The different domain models for a product variant are derived by applying a subset of the deltas of the respective domain.

Setting up the Consistency Preserving Environment. The consistency preserving environment consists of the meta-models of the different domains between which variability is propagated and a definition of incremental consistency preservation connecting these domains. In [20] the Vitruvius approach [13] and its reference implementation Vitruv[3] are proposed, which we continue to use. Therefore, for the BCS, the consistency preserving environment consists of the two domain meta-models and the CPRs. The statemachine meta-model **1** includes 8 classes: a `Statemachine` as root element. `States`, `Transitions` and `CommunicationSignals` as main elements. `Regions` define separate parallel behaviors or nested behaviors within a `State`. In addition, `Transitions` can reference `Triggers`, `Behaviors` and `Constraints`, of which the last two refer to received and sent `CommunicationSignals`. The component meta-model **2** consists of 5 classes: a `System` as root element, `Components` which can have subcomponents and contain `Ports` via which `Signals` are sent and received and `Connectors` to define communication paths between components. There are two sets of CPRs: First, the Statemachine-to-Component **3**, for which 8 reactions and 8 routines are defined. These include for example the reaction `SignalInsertedIntoTrigger` and the routine `createAndInsertInputPort`. They ensure that if a `Transition` in a statemachine `Region` is triggered by a certain `CommunicationSignal`, then the `Component` with that behavior has to have an input `Port` that can receive the corresponding `Signal` since otherwise this transition would be "dead". This is ensured by checking whether a matching `Port` is already present and, if not, it is inserted into the `Component` which corresponds to the `Transition`'s `Statemachine`-container. Second, the Component-to-Statemachine-CPRs **4**, for which 6 reactions and 6 routines are defined. For example, the reaction `SignalInsertedIntoSystem` and the routine `createAndInsertCommunicationSignal` which together ensure that for each new `Signal` in the component model, a corresponding `CommunicationSignal` in the statemachine model is created.

Starting from a Consistent Initial Model. In the consistency preserving environment, an initial pair of consistent models (a V-SUM) is created. It consists of one statemachine model **5** and one component model **6**. As example for this initial VSUM we choose the product variant P0 from the BCS report [16] which is also shown in Fig. 1. The component model of this product variant consists of 4 different `Components`: an `Exterior Mirror (EM)`, a `Manual Power Window (ManPW)`, a `Finger Protection (FP)`, and a `Human Machine Interface (HMI)`, which are described in detail in [16, 20]. The statemachine model consists of 4 corresponding `Regions`, one for each `Component`, which include multiple `States` and `Transitions` describing their behavior.

Translating and Applying Source Domain Variability. In the consistency preserving environment, a delta is applied **7** to the initial source domain model (`P0.statemachine`) **5**. In the running example, we model the delta to add the feature *Heatable* to heat the exterior mirrors to free them from ice in the winter. The statemachine delta adds `States`, `Transitions`, `CommunicationSignals`, `Triggers` and `Behaviors` to activate the heating if the mirror gets too cold and deactivate the heating after a certain time period is elapsed. Applying this delta derives a new source domain model variant (`P17.statemachine`) **8**.

Propagating and Retrieving Target Domain Variability. Applying the source domain delta will trigger the reactions in the CPRs of the corresponding direction **3**. For example, adding a `CommunicationSignal` to a `Transition`'s `Trigger` will activate the reaction `SignalInsertedIntoTrigger` and the routine `createAndInsertInputPort` which will then add a new input `Port` into the `EM Component` of the initial target domain model variant (`P0.component`) **6**. This will derive a new target domain model variant (`P17.component`) **9**, which forms a new V-SUM with the new source domain model

[3] https://github.com/vitruv-tools

variant 8. The consequential change is persisted as target domain delta representing the propagated variability 10.

4 Implementation

To evaluate the concept of variability propagation outlined in Section 3, we implement a prototype based on the *Eclipse Modeling Framework* [30] which is an often used reference implementation of the MOF Standard of the *Object Management Group* [22]. This prototype is a combination of Eclipse Plug-Ins and Maven Modules implemented in Java. It consists of six main components, explained below. The generic delta meta-model, the generation of domain-specific delta dialects and the delta interpreter are inspired by `DeltaEcore` [28] and adapted to conform to the Ecore standard.

BCS Meta-Models. To process the BCS as case study with our implementation, it has to be represented in an ecore-conform format. Therefore, we create two meta-models, one for each BCS-Domain: components and statemachines. The meta-model designs have been described in Section 3. Using these meta-models, model code is generated as well as editor code with which BCS models can be developed in an Eclipse Plug-In editor.

Generic Delta Meta-Model. Following the description of delta modeling from [25, 26] we create a domain-independent meta-model for `Deltas`. This meta-model defines what kinds of delta operations exist for ecore-conform models. For example, it defines the delta operation `AddToMultiValuedReference`, which adds a value to a specified `EReference` that can be contained in any class or interface of a meta-model. `Deltas` contain a list of delta operations and are themselves contained in a `DeltaRepository`.

Domain-Specific Delta Dialect Generator. To make delta modeling easier approachable, we implement a generator which can generate a domain-specific, so-called *delta dialect* from a meta-model. For each existing `EClass`, `EReference`, and `EAttribute`, the applicable delta operations from the generic delta meta-model are extended with new classes. These act mainly as syntactic sugar but can also be manually enriched with additional information if needed. For example, in the statemachine meta-model, `Transitions` are contained in the multi-valued `EReference` `Region::transitions` and they themselves contain the single-valued non-containment reference `Transition::sourceState`. The delta dialect contains, e.g., the delta operations `CreateAndAddTransitionIntoRegion` and `SetSourceStateInTransition`. Since the delta dialect is a meta-model, EMF can be used to generate model code and editors for them as well, allowing to create deltas with a user interface. These deltas can be persisted and loaded in the XMI format [21] to enable development for variant-rich systems.

Delta Interpreter. In addition to their delta operations, `Deltas` contain an *applicability condition* and *after-clause*. Applicability conditions state the conditions which have to be fulfilled to select this delta for application, e.g., features of a product line have to be (not) part of a product configuration to apply this delta. After-clauses express a partial order on deltas which have to be followed during product derivation (cf. Section 2). Using this information, a subset of available deltas can be selected and executed by executing all contained delta operations. The execution of delta operations accepts a model and returns an updated model. The final model is saved as the derived model variant.

Delta-Change Translator. To apply delta in the consistency preserving environment (cf. Section 3), they have to be translated into a representation that Vitruv can work with. Vitruv defines an own meta-model for *EChange*s which are similar to `Deltas`. Translating forth and back between both notions of change is possible without information loss and realized using *Epsilon*[4], a model-to-model transformation framework.

Vitruv-Integration. So far, the `EChange` meta-model was only used as internal data structure. We implement an integration of Vitruv to load persisted `EChanges` and apply them as original changes (cf. Section 2) in a V-SUM. With this integration we can retrieve the consequential `EChanges` and persist them to subsequently translate them back to `Deltas`.

5 Evaluation

The guiding question of this evaluation is whether the proposed combination of incremental consistency preservation and delta-oriented variability modeling *(variability propagation)* enhances multi-domain variant-rich systems development. This question addresses the need to evaluate in practice the conceptual integration proposed in [20]. We implemented a prototype and apply it to a representative multi-domain case study to propagate variability across domains. The evaluation considers the synthesized variability artifacts with respect to soundness and completeness.

5.1 Research Questions

To evaluate the proposed variability propagation approach in more detail, we propagate the deltas of a delta-oriented product line. We decompose the evaluation into the following research questions.

RQ1 — Soundness: To what extent is the variability synthesized by variability propagation correct with respect to the expected variability of a target domain? To avoid an additional (possibly manual) validation of variability artifacts synthesized by variability propagation, the *soundness* of variability propagation should be as high as possible. We assess the *soundness* of our implementation in two ways. **RQ1.1** *Delta Soundness:* First, by inspecting for each synthesized delta if it — i.e., its delta operations — express only variability that is also modeled in the ground truth. This can be achieved by comparing types (add/ remove), values and structure (order of operations) between synthesized delta operations and those of the ground truth target domain deltas. **RQ1.2** *Model Soundness:* Second, we inspect whether the application of the synthesized delta and the ground truth target domain delta result in the same model variant. This is achieved by comparing the respective target domain models regarding presence and structure of model elements. As variability propagation is entirely guided by explicitly defined consistency preservation rules (CPRs), we expect high soundness.

RQ2 — Completeness: To what extent does variability propagation synthesize the expected variability of a target domain? Variability propagation aims to reduce manual effort in multi-domain variability modeling. Therefore, the usefulness of this approach depends

[4] https://github.com/eclipse-epsilon/epsilon

on the *completeness* of the synthesized deltas. This is assessed in two ways. **RQ2.1** *Delta Completeness:* First, by assessing for how many expected ground truth delta operations a correct (see RQ1) delta operation was actually synthesized. **RQ2.2** *Model Completeness:* Second, both the synthesized and the ground truth deltas are applied to derive the respective model variant. The *completeness* is then assessed by checking for how many ground truth model elements a correct synthesized model element was derived. We expect variability propagation to have a higher *soundness* than *completeness*.

RQ3 — CPR Coverage: To what extent do consistency preservation rules determine target domain model elements from source domain model elements? Since variability propagation exploits the incremental consistency preservation rules (CPRs), understanding their influence on how much variability can be synthesized is important. We evaluate this influence by inspecting the *coverage* of the CPRs — i.e., counting for both propagation directions (statemachine → component, component → statemachine) how many target domain elements can be determined from source domain elements via CPRs.

5.2 Evaluation Setup

The proposed variability propagation approach is evaluated using the setup shown in Fig. 2, which illustrates the evaluation with Vitruv. A single execution of the evaluation process is referred to as a *VPR*. A VPR starts with the selection of source domain deltas 7 which are to be propagated. To be applicable, precondition deltas have to be executed first to create the initial source model 5. The source domain deltas 7 are translated to the notion of change used in Vitruv, so-called changes 11. Applying these source domain changes triggers the CPRs 3, 4 which apply a *consequential change* 12 in the target domain. To apply these consequential changes outside Vitruv, they are translated back to delta operations and combined into a single delta, which we call a *synthesized delta* 10.

The evaluation of variability propagation consists of two steps: For the first step (RQ1.1, RQ2.1), we compare 14 the *synthesized delta* 10 with its corresponding *ground truth delta* 13. Since deltas may differ in number and order of their delta operations, matching them is based on type and name equality. This matching is extended iteratively using the identifiers of already matched operations. We call the delta operations from *synthesized delta* that can be matched *correct operations*. Differences in the number or type of delta operations, however, do not necessarily imply differences in the resulting model variants. Distinct delta operations may produce equivalent models. Therefore, in the second step (RQ1.2, RQ2.2), we compare the derived target domain models in addition to the deltas. To prepare the derivation of these models, in the target domain an *initial target model* 15 is derived. Applying the *synthesized delta* and the *ground truth delta* to the *initial target model* derives the *synthesized model* 16 and the *ground truth model* 17 respectively. The resulting models are compared 18. Only the model elements which were added by applying the deltas to the *initial target model* are considered. This focuses the comparison on the propagated variability and excludes elements which were already present in the *initial target model*. These added model elements are matched based on type and name equality. Matched elements of the *synthesized model* are called *correct element*.

For **RQ1.1**, we calculate the *soundness* as the ratio between the number of *correct operations* and the total number of synthesized delta operations and express it in percentage. For **RQ1.2**, the *soundness* is calculated as the ratio between the number of *correct element* and the number of added model elements in the *synthesized model*. In both cases, *soundness* can reach from 0 (“None of the synthesized operations was in the *ground truth delta* “) up to 1 (“Every synthesized operation was also in the *ground truth delta* “).

For **RQ2.1**, we calculate the *completeness* as the ratio between the number of *correct operations* and the total number of ground truth delta operations. For **RQ2.2**, the *completeness* is calculated as the ratio between the number of *correct element* and the number of added model elements in the *synthesized model*. In both cases, *completeness* can reach from 0 (“None of the expected operations was synthesized“) to 1 (“All expected operations were synthesized“).

For **RQ3**, we count for how many of the target domain metamodel classes instances can be created or deleted and how many structural features can be modified. For each propagation direction, we qualitatively analyze possible influences between *coverage*, and *soundness* and *completeness*.

5.3 Subject System

In Section 3, we introduced variability propagation using the BCS and provided an example of how one product variant being transformed into another using deltas. We continue to use the BCS as the subject system. In total, the BCS enables the derivation of 11,616 product variants based on 39 statemachine deltas and 34 component deltas, which comprise a total of 3785 and 1140 delta operations, respectively.

In Section 5.2, we described our evaluation setup and how the propagation of a delta looks like. To evaluate our implementation, we execute multiple *variability propagation runs*, short VPRs, each defined by the following information:

- source domain deltas forming the initial source model 5
- one or more source domain deltas for propagation 7
- one or more target domain ground truth deltas 13
- source domain deltas forming the initial target model 15

In total, we define 75 VPRs for our evaluation: 41 VPRs for the variability propagation from statemachine to component domain, shown in Table 1 and 34 VPRs for the variability propagation from component to statemachine domain, shown in Table 2. For both propagation directions, the VPRs for the evaluation are manually constructed according to the following procedure:

① For each source domain delta, we created at least one VPR. ② We determined the deltas to construct the initial source model by transitively collecting all required deltas from the source domain deltas's *after-clauses* (cf. Section 2). ③ The expected target domain deltas were manually identified by comparing the *application conditions* (cf. Section 2) of source and target domain deltas. ④ If a one-to-one correspondence between source and target domain delta is impossible due to structural differences in the decomposition of deltas, multiple deltas are grouped and propagated/ attempted to be synthesized together. The grouping aligns semantically equivalent

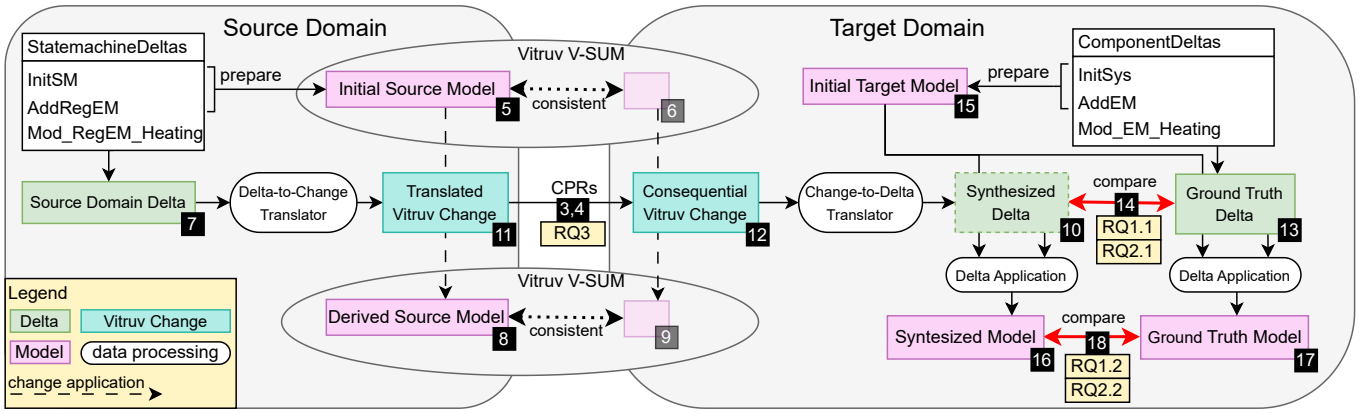


Figure 2: Evaluation Procedure of Consistency Preserving Variability Propagation. The figure extends Fig. 1 and illustrates how source domain deltas are propagated via Vitruv changes and translated back into deltas, enabling comparison of synthesized and ground truth deltas as well as derived models.

variability across both domains, which ensures that each VPR compares corresponding variability rather than individual deltas. ⑤ Finally, additional VPRs were added for *context-sensitive* deltas, i.e., cases in which propagating the same source domain delta synthesizes different target domain deltas depending on the initial source model. Such cases are identified by inspecting target domain deltas whose after-clause references at least two predecessor deltas which do not require each other in their after-clauses. For each predecessor delta, a separate VPR is created in which this predecessor delta is propagated after the remaining predecessor deltas are already part of the initial source model.

To illustrate the steps ① ② ③, consider VPR #21 in Table 1, which is used as example in Section 3 and in Fig. 2. The statemachine delta `Mod_RegEM_Heating` can only be applied after the delta `AddRegEM` which itself requires the delta `InitSM`. The corresponding target domain delta in the component domain is `Mod_EM_Heating` and requires the delta `AddEM` which itself requires the delta `InitSys`.

To illustrate the step ④, consider VPR #22 in Table 1: Here, one target domain delta in the component domain expresses the same variability as five statemachine source domain deltas together. While in #22 in Table 1 the five statemachine deltas modify one existing and add four new `Regions`, the component delta `DLEDEM` modifies the existing corresponding `Component` and adds all four new `Components` at once. This structural difference between deltas enforces the combined propagation in a single VPR or in the opposite direction (see #10 in Table 2) the combined synthesis.

To illustrate the step ⑤, consider the VPRs #2, #3, #5, and #7 in Table 1: In #2 and #5 the delta `AddRegEM` is propagated, whereas in #3 and #7 the delta `AddHMI_Core` is propagated. Both deltas are independent of each other. After the VPRs #2 and #3 are already defined, we aim to synthesize `Wire_HMI_EM` which references both target domain deltas, `AddHMI` and `AddEM`, in its after-clause. This can be achieved by propagating the source domain delta `AddRegEM` when `AddHMI_Core` is already part of the initial source model. This constitutes the VPR #5. It can also be achieved by propagating the source domain delta `AddHMI_Core` when `AddRegEM` is already part of the initial source model. This constitutes the VPR #7. Internally, this behavior is caused by the model elements affected by the propagated deltas. In #5, the statemachine `Region` for the "Exterior Mirror" is added which, when propagated, synthesizes the addition of the `Component` "Exterior Mirror". Since the `Component` "Human Machine Interface" is already present in the target domain model and both components can communicate with each other via `Ports` with the same `Signal`, their `Ports` are connected with `Connectors`. This addition of `Connectors` is the additional variability expressed in the target domain delta `Wire_HMI_EM`.

Following this procedure to construct VPRs, the whole variability of the source domain is propagated and every context-sensitive delta is propagated once in every context in which a context-sensitive effect occurs. Also, the whole variability of the target domain is attempted to be synthesized and therefore all context-sensitive effects of the target domain are also attempted to be synthesized once. Defining these more fine-grained VPRs for deltas rather than complete product variants, reduces the number of needed VPRs while the propagation of complete product variants can still be realized by combining multiple VPRs.

5.4 Evaluation Results

RQ1 — Soundness. We executed 75 VPRs to assess the soundness of synthesized delta operations and derived model variants. For **RQ1.1**, the results for the 41 VPRs from statemachine domain to component domain are shown in Table 1. In all VPRs, every synthesized delta operation could be matched to a corresponding delta operation in the ground truth resulting in delta soundness of 100%. For **RQ1.2**, applying the synthesized deltas derived model variants whose added model elements matched the respective ground truth fully, resulting in a model soundness of 100%.

The results of the 34 VPRs from component domain to statemachine domain are shown in Table 2. Here, also all synthesized delta operations could be matched, resulting in a delta soundness of 100% for **RQ1.1**. Furthermore, also all derived model elements were matched, resulting in a model soundness of 100% **RQ1.2**.

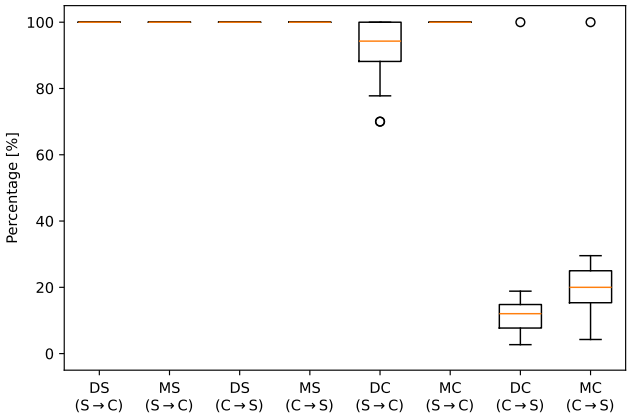


Figure 3: Evaluation Results Boxplots. The figure shows the distribution of delta soundness (DS), model soundness (MS), delta completeness (DC), and model completeness (MC) expressed in percent for the two variability propagation directions: statemachine to component (S→C) and component to statemachine (C→S).

RQ2 — Completeness. For the propagation from statemachine domain to component domain, delta completeness (**RQ2.1**) is 100% in 17 out of 41 VPRs, while in the remaining 24 VPRs expected delta operations were not synthesized completely, ranging from 70% to 96.97% (cf. Table 1). Despite this, model completeness (**RQ2.1**) is 100% across all VPRs, meaning that all expected model elements could be derived by applying the synthesized deltas. For the propagation from component domain to statemachine domain, delta completeness is substantially lower. Only 1 out of 34 synthesized deltas contain all expected delta operations, while the remaining VPRs range between 2.7% and 18.84% delta completeness (cf. Table 2). This reduction also affects model completeness, which ranges from 4.26% to 29.55% across all VPRs.

The distribution of the metrics concerning soundness and completeness are shown in Fig. 3. The first 4 plots illustrate the always perfect soundness of variability propagation. For completeness, the substantially lower measurements for the propagation direction from component domain to statemachine domain are shown as well as the fact that incomplete deltas can still lead to complete models.

RQ3 — CPR Coverage. To evaluate how strongly the two modeling domains determine each other, we analyze the consistency preservation rules (CPRs) in both propagation directions in terms of how many meta-model elements of the respective target domain are covered by CPRs. In the statemachine-to-component direction, the CPRs enable the creation of all 5 entity types in the component domain and the manipulation of 13 out of 14 structural features (references and attributes) in the component meta-model. In the opposite direction, from component to statemachine, the CPRs enable the creation of 3 out of 8 entity types in the statemachine domain and allow manipulation of 5 out of 25 structural features in the statemachine meta-model. Overall, CPR coverage is significantly higher in the statemachine-to-component direction.

5.5 Discussion

RQ1 — Soundness. The 100% delta and model soundness indicate that variability propagation does not synthesize unwanted variability in the target domain. This is consistent with the approach's design, since variability is generated exclusively via consistency preservation rules (CPRs). Thus, propagated variability is limited to consistency-preserving changes. Although soundness is expected by design, it is not guaranteed, as mismatches between propagated variability and ground truth may still yield unsound results. The evaluation therefore confirms that the implementation correctly realizes the CPR-based propagation for the case study. Consequently, no error-correction step is required after variability is propagated. However, synthesized deltas resulting from the propagation of multiple deltas at once may be refactored in a post-processing step.

RQ2 — Completeness. The results show that variability propagation can synthesize target domain variability completely or partially, depending on the propagation direction. While delta completeness is often less than 100%, this does not necessarily prevent the derivation of complete model variants. In the statemachine-to-component direction, incomplete synthesized deltas can still derive complete model variants. This can be due to redundancy in deltas and structural differences between deltas across domains. For example, in VPR number 12, missing delta operations creating two `Signals` are already included in another delta. After delta application, the model variant remains complete despite the incomplete synthesized delta. This case shows that redundancy and delta structure differences can lead to incompleteness. However, this effect does not generalize to all cases of incomplete deltas. In the component-to-statemachine direction, low delta completeness leads to low model completeness. This is because the statemachine domain contains behavioral constructs (e.g., `State`, `Transition`), that cannot be inferred from the component domain's structural constructs. However, incomplete synthesized deltas still contain all consistency-critical delta operations as they are synthesized by CPRs. This ensures that the resulting model variants remain consistent. In summary, variability propagation does not always synthesize complete deltas, but rather a consistency-preserving subset.

RQ3 — CPR Coverage. For statemachine to component domain, CPRs provide full coverage of component meta-model class instantiation. Only the reference `subComponents`, for nesting `Components` is not covered by CPRs. However, this reference is not used in the BCS case study. Therefore the CPRs in this propagation direction provide full coverage. For component to statemachine domain, CPRs cover 3/8 entity types and 5/25 structural features. Neglecting unused meta-model elements, the CPRs cover 3/7 entity types and 5/19 structural features, which still restricts variability propagation. CPR coverage defines an upper bound on what can be synthesized, but does not guarantee that all reachable elements are always correctly synthesized. The results for completeness indicate a strong asymmetry between the two propagation directions. This asymmetry is directly reflected in the structure and coverage of the corresponding consistency preservation rules (CPRs). Therefore, the case study shows that CPR coverage has a strong practical impact on the achievable completeness and explains the differences in the results between propagation directions.

5.6 Threats to Validity

Internal Validity. The implementation of variability propagation as well as the manually defined evaluation artifacts (ground-truth deltas and the 75 VPRs) may contain errors that influence the shown results. To mitigate this threat, the implementation was incrementally tested during development using the parts of the subject system. Intermediate results were manually inspected by us and coworkers. Underlaying artifacts such as the BCS meta-models, their *delta dialects* and the *delta interpreter* (cf. Section 4) were also used in other research projects and student theses in parallel and showed no errors. The deltas of the BCS case study were partially refactored to match pure delta modeling [27] and correct some mistakes in the original artifacts. These were inspected together with coworkers, who also further worked with these artifacts. The VPRs were defined following the described procedure, reviewed by coworkers and inspected in detail during the incremental testing of the implementation.

The translation between deltas and changes poses the threat of information loss between these different notions of change. However, a prior evaluation conducted in the context of a bachelor's thesis [29] tested this translation and showed that the expressiveness of both notions is equivalent.

External Validity. As case study, we chose the full BCS as described in [16] and transformed it into pure delta modeling [27]. Since the evaluation does not focus on derived product variants, we did not cover them but rather ensured that all defined deltas of both domains are propagated and targeted by propagation (cf. Section 5.3). The BCS does not represent all possible variant-rich systems/ product lines, but it is a good representation for a model product line. The BCS has been used in multiple studies before [15, 31]. For this study, we also used an Ecore-conform [22] implementation of the BCS. Ecore is a widely used modeling standard. It is the required format for Vitruv and the used delta dialects. Concerning this standard, the only shortcoming of the BCS is that it does not use all sophisticated constructs of Ecore like inheritance and generics. The BCS consists only of two domains. Real-world product line may contain more than two domains resulting in multiple special cases like different propagation paths from a source domain through multiple intermediate domains to the target domain. Possible resulting conflicts were not subject to our investigation. However, when writing CPRs this threat can be mitigated by careful testing. This additional work occurs only once per new domain but can then be reused many times during variability propagation.

6 Related Work

Multi-Domain Consistency Preservation for Single Products. Previous work [20] which envisioned the approach which we evaluate, builds on the *Vitruvius* approach [13]. The *Vitruvius* approach introduces the V-SUM in which connects multiple, possibly already existing, metamodels with each other using CPRs. This enables the representation of multi-domain products as loosely-coupled single models updating each other during development. This fits well with our goal of establishing variability of a multi-domain system as separate variability artifacts. Another, more exploratory approach is *DesignSpace* [9, 19] which does not use imperative consistency rules, but generates multiple consistent options to preserve consistency from which a developer can choose. While this increases flexibility, it requires coordination between engineers of different domains, which we aim to reduce through automated variability propagation. These approaches do not address the consistent development of variant-rich multi-domain systems.

Consistency Development of Variant-Rich Systems. Consistency preservation becomes more complex, when variant-rich systems, instead of single products, must be kept consistent. New consistency types must be considered. *Incremental Consistency Checking* [14] classified additional consistency types, used in [11]. While our variability propagation considers these consistency types, we also utilize them to improve the development of variant-rich systems. *VaVe* [3] focuses on maintaining consistency between variants and versions of artifacts, but is limited on a single domain and the tracking of internal subcomponents of a model. In contrast, we establish consistency on deltas as primary variability artifacts for multiple domains allowing for more reuse. Greiner et al. [12] proposes another form of variability propagation in which feature annotations of variability artifacts from different domains are kept consistent instead of propagating the artifacts themselves as we do. The *Virtual Platform* [4, 17, 18] supports *platform-oriented development* [1, 2] by reconstructing and maintaining a virtual product-line platform from existing product variants. However, it also focuses on a single domain while we establish consistent variability artifacts for multiple domains. [7] lifted the consistency preservation rules (CPRs) of the *Vitruvius* approach [13] from a single product to a delta-oriented product line, supporting platform-oriented development. While this approach maintains consistency between different domains of a product line, we propagate the variability of one domain to the other to create a consistent multi-domain variant-rich system.

7 Conclusion

In this paper, we introduced and evaluated a prototypical implementation of *variability propagation* as proposed in [20]. Consistency preservation rules can effectively be exploited to synthesize deltas in variant-rich multi-domain systems. Synthesized deltas are sound but may be incomplete, depending on the domains and CPRs used for variability propagation. Despite incomplete deltas, complete model variants can still be derived in some cases due to structural differences between domains. Overall, the evaluation demonstrates that variability propagation is a practical mechanism to enhance the development of multi-domain variant-rich systems.

Beyond the empirical findings, variability propagation is a first step towards *platform-oriented development* [1, 2] of multi-domain variant-rich systems. Our approach enables the initialization of a variability aware V-SUM (Var-V-SUM) [10]. Future work will extend support for platform-oriented development by enabling consistency-preserving refactoring of propagated deltas and investigate the applicability of variability propagation in *product-oriented development* [1, 2] where newly developed variability can be propagated across domains.

Acknowledgments

This work was supported by funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1608 - 501798263 and the KASTEL Security Research Labs.

References

[1] Sofia Ananieva. 2022. *Consistent View-Based Management of Variability in Space and Time.* Ph. D. Dissertation. Karlsruhe Institute of Technology. doi:10.5445/IR/1000148819

[2] Sofia Ananieva, Sandra Greiner, Jacob Krueger, Lukas Linsbauer, Sten Gruener, Timo Kehrer, Thomas Kuehn, Christoph Seidl, and Ralf Reussner. 2022. Unified Operations for Variability in Space and Time. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS '22)*. 1–10. doi:10.1145/3510466.3510483

[3] 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 (VAMOS '18)*. 3–10. doi:10.1145/3168365.3168377

[4] 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 (ICSE Companion 2014)*. 532–535. doi:10.1145/2591062.2591126

[5] 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*. 206–219. doi:10.1007/978-3-642-14819-4_15

[6] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *Proceedings of the 7th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS '13)*. 1–8. doi:10.1145/2430502.2430513

[7] Simon Bothe. 2026. Enabling Cross-Domain Consistency Preservation for Delta-Oriented Product Line Development. doi:10.5445/IR/1000195066

[8] K. Czarnecki and S. Helsen. 2006. Feature-Based Survey of Model Transformation Approaches. 45, 3 (2006), 621–645. doi:10.1147/sj.453.0621

[9] Andreas Demuth, Markus Riedl-Ehrenleitner, Alexander Nöhrer, Peter Hehenberger, Klaus Zeman, and Alexander Egyed. 2015. DesignSpace: An Infrastructure for Multi-User/Multi-Tool Engineering. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15)*. 1486–1491. doi:10.1145/2695664.2695697

[10] Fabian Eger, Thomas Weber, Lars König, Dirk Neumann, Kevin Feichtinger, Erik Burger, and Ralf Reussner. 2026. Towards Consistent, Variability-Aware Views in Systems Development. accepted for publication, to appear.

[11] Hafiyyan Sayyid Fadhlillah, Sandra Greiner, Kevin Feichtinger, Rick Rabiser, and Alois Zoitl. 2024. Managing Variability of Cyber-Physical Production Systems: Towards Consistency Management. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24)*. 945–949. doi:10.1145/3652620.3688216

[12] Sandra Greiner, Michael Nieke, and Christoph Seidl. 2022. Towards Trace-Based Synchronization of Variability Annotations in Evolving Model-Driven Product Lines. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems* (New York, NY, USA, 2022-02-23) *(VaMoS '22)*. Association for Computing Machinery, 1–10. doi:10.1145/3510466.3510470

[13] 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 (2021), 110815. doi:10.1016/j.jss.2020.110815

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

[15] Remo Lachmann, Sascha Lity, Mustafa Al-Hajjaji, Franz Fürchtegott, and Ina Schaefer. 2016. Fine-Grained Test Case Prioritization for Integration Testing of Delta-Oriented Software Product Lines. In *Proceedings of the 7th International Workshop on Feature-Oriented Software Development* (New York, NY, USA, 2016-10-30) *(FOSD 2016)*. Association for Computing Machinery, 1–10. doi:10.1145/3001867.3001868

[16] Sascha Lity, Remo Lachmann, Malte Lochau, and Ina Schaefer. 2013. Delta-Oriented Software Product Line Test Models - The Body Comfort System Case Study. *Technical report, TU Braunschweig* (2013).

[17] Wardah Mahmood, Daniel Strüber, Thorsten Berger, Ralf Lämmel, and Mukelabai Mukelabai. 2021. Seamless Variability Management with the Virtual Platform. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1658–1670. doi:10.1109/ICSE43902.2021.00147

[18] Wardah Mahmood, Gül Çalıklı, Daniel Strüber, Ralf Lämmel, Mukelabai Mukelabai, and Thorsten Berger. 2024. Virtual Platform: Effective and Seamless Variability Management for Software Systems. 50, 11 (2024), 2753–2785. doi:10.1109/TSE.2024.3406224

[19] Luciano Marchezan, Wesley K. G. Assuncao, Roland Kretschmer, and Alexander Egyed. 2022. Change-Oriented Repair Propagation. In *Proceedings of the International Conference on Software and System Processes and International Conference on Global Software Engineering (ICSSP '22)*. 82–92. doi:10.1145/3529320.3529330

[20] 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* (A Coruña) *(SPLC-A '25)*. Association for Computing Machinery, New York, NY, USA, 88–93. doi:10.1145/3744915.3748466

[21] Object Management Group (OMG). 2015. *XML Metadata Interchange (XMI) Specification Version 2.5.1 (formal/2015-06-07)*. Meta Object Facility.

[22] Object Management Group (OMG). 2019. *Meta Object Facility (MOF) Core Specification Version 2.5.1 (formal/2019-10-01)*. Meta Object Facility.

[23] Klaus Pohl, Günter Böckle, and Frank Linden. 2005. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Vol. 1. doi:10.1007/3-540-28901-1

[24] 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)*. 55–71. doi:10.1109/FOSE.2007.22

[25] Ina Schaefer. 2010. Variability Modelling for Model-Driven Development of Software Product Lines. In *Fourth International Workshop on Variability Modelling of Software-Intensive Systems (ICB-Research Report, Vol. 37)*. 85–92. http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf

[26] 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 - 14th International Conference (Lecture Notes in Computer Science, Vol. 6287)*. 77–91. doi:10.1007/978-3-642-15579-6_6

[27] Ina Schaefer and Ferruccio Damiani. 2010. Pure Delta-Oriented Programming. In *Proceedings of the 2nd International Workshop on Feature-Oriented Software Development* (New York, NY, USA, 2010-10-10) *(FOSD '10)*. Association for Computing Machinery, 49–56. doi:10.1145/1868688.1868696

[28] Christoph Seidl, Ina Schaefer, and Uwe Aßmann. 2014. DeltaEcore-A Model-Based Delta Language Generation Framework, Vol. P-225. 81–96. https://dl.gi.de/handle/20.500.12116/20956

[29] Darius Seiter. 2025. Comparing and Translating Notions of Change for Delta-Oriented Variability. doi:10.5445/IR/1000186968

[30] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: Eclipse Modeling Framework*. Pearson Education.

[31] David Wille, Tobias Runge, Christoph Seidl, and Sandro Schulze. 2017. Extractive Software Product Line Engineering Using Model-Based Delta Module Generation. In *Proceedings of the 11th International Workshop on Variability Modelling of Software-Intensive Systems* (New York, NY, USA, 2017-02-01) *(VaMoS '17)*. Association for Computing Machinery, 36–43. doi:10.1145/3023956.3023957

#	Source Statemachine Delta	Initial Source Model	Target Component Delta	Initial Target Model	RQ1.1	RQ2.1	RQ1.2	RQ2.2
1	InitSM		InitSys		100	100	100	100
2	AddRegEM	InitSM	AddEM	InitSys	100	100	100	100
3	AddHMI_Core	InitSM	AddHMI	InitSys	100	100	100	100
4	AddFP	InitSM	AddFP	InitSys	100	100	100	100
5	AddRegEM	InitSM, AddHMI_Core	AddEM, Wire_HMI_EM	InitSys, AddHMI	100	89.74	100	100
6	AddFP	InitSM, AddHMI_Core	AddFP, Wire_HMI_FP	InitSys, AddHMI	100	92.31	100	100
7	AddHMI_Core	InitSM, AddRegEM	AddHMI, Wire_HMI_EM	InitSys, AddEM	100	89.74	100	100
8	AddHMI_Core	InitSM, AddFP	AddHMI, Wire_HMI_FP	InitSys, AddFP	100	96.97	100	100
9	AddAutoPW	InitSM	AddAutoPW	InitSys	100	100	100	100
10	AddFPinAutoPW	InitSM, AddAutoPW	AddFPinAutoPW	InitSys, AddAutoPW	100	100	100	100
11	AddFP	InitSM, AddAutoPW, AddFPinAutoPW	AddFP, Wire_FP_AutoPW	InitSys, AddAutoPW, AddFPinAutoPW	100	80.00	100	100
12	AddFPinAutoPW	InitSM, AddFP, AddAutoPW	AddFPinAutoPW, Wire_FP_AutoPW	InitSys, AddFP, AddAutoPW	100	77.78	100	100
13	AddAutoPW	InitSM, AddHMI_Core	AddAutoPW, Wire_HMI_AutoPW	InitSys, AddHMI	100	91.11	100	100
14	AddHMI_Core	InitSM, AddAutoPW	AddHMI, Wire_HMI_AutoPW	InitSys, AddAutoPW	100	94.29	100	100
15	AddManPW	InitSM	AddManPW	InitSys	100	100	100	100
16	AddFPinManPW	InitSM, AddManPW	AddFPinManPW	InitSys, AddManPW	100	100	100	100
17	AddFP	InitSM, AddManPW, AddFPinManPW	AddFP, Wire_FP_ManPW	InitSys, AddManPW, AddFPinManPW	100	80.00	100	100
18	AddFPinManPW	InitSM, AddFP, AddManPW	AddFPinManPW, Wire_FP_ManPW	InitSys, AddFP, AddManPW	100	77.78	100	100
19	AddManPW	InitSM, AddHMI_Core	AddManPW, Wire_HMI_ManPW	InitSys, AddHMI	100	90.00	100	100
20	AddHMI_Core	InitSM, AddManPW	AddHMI, Wire_HMI_ManPW	InitSys, AddManPW	100	94.29	100	100
21	Mod_RegEM_Heating	InitSM, AddRegEM	Mod_EM_Heating	InitSys, AddEM	100	100	100	100
22	DAddEMLEDEM, EMLedTop, EMLedLeft, EMLedBottom, EMLedRight	InitSM, AddRegEM	DLEDEM	InitSys, AddEM	100	100	100	100
23	AddLED_EMH	InitSM, AddRegEM, Mod_RegEM_Heating	AddLED_Heatable	InitSys, AddEM, Mod_EM_Heating	100	81.25	100	100
24	AddFP_LED	InitSM, AddFP	DLEDFingerProtection	InitSys, AddFP	100	92.86	100	100
25	AddManPW_LED, DAddHMILEDManPW	InitSM, AddHMI_Core, AddManPW	DLEDManPW	InitSys, AddManPW, AddHMI	100	96.67	100	100
26	AddAutoPW_LED	InitSM, AddAutoPW	DLEDAPW	InitSys, AddAutoPW	100	93.48	100	100
27	AddCLS	InitSM	DCLS	InitSys	100	100	100	100
28	DAddManPWCLS	InitSM, AddManPW, AddFPinManPW, AddCLS	WireCLSManPW	InitSys, AddManPW, DCLS	100	70.00	100	100
29	DAddAutoPWCLS	InitSM, AddAutoPW, AddFPinAutoPW, AddCLS	WireCLSAutoPW	InitSys, AddAutoPW, DCLS	100	70.00	100	100
30	DAddCLSAL	InitSM, AddCLS	DAL	InitSys, DCLS	100	100	100	100
31	AddRCK, DAddCLSRCK	InitSM, AddCLS	DRCK	InitSys, DCLS	100	100	100	100
32	AddLEDCLS	InitSM, AddCLS	DLEDCLS	InitSys, DCLS	100	81.25	100	100
33	DAddRCKSF	InitSM, AddRCK	DRCKSF	InitSys, DCLS, DRCK	100	100	100	100
34	AddAS, DAddHMIAS	InitSM, AddHMI_Core	DAS	InitSys, AddHMI	100	100	100	100
35	DAddASCAS	InitSM, AddCLS, AddRCK, AddAS	DCAS	InitSys, DCLS, DRCK, AddHMI, DAS	100	77.78	100	100
36	AddAutoPW_LED, DAddLEDAutoPWCLS	InitSM, AddAutoPW, AddCLS	DLEDAPWCLS	InitSys, AddAutoPW, DCLS, WireCLSAutoPW	100	88.16	100	100
37	DAddASIM	InitSM, AddAS	DASIM	InitSys, AddHMI, DAS	100	100	100	100
38	DAddASIM, LedASIM	InitSM, AddAS	DASIM, DLEDASIM	InitSys, AddHMI, DAS	100	100	100	100
39	DAddHMILEDAS, LedASAD, LedASAL, LedASAC	InitSM, AddHMI_Core, AddAS	DLEDAS	InitSys, AddHMI, DAS	100	87.38	100	100
40	AddFP, DAddRCKCAP	InitSM, AddAutoPW, AddRCK	AddFP, DRCKCAP	InitSys, DCLS, DRCK, AddAutoPW	100	92.31	100	100
41	AddFP, DAddRCKCAP, DAddRCKCAPSF	InitSM, AddAutoPW, AddRCK, DAddRCKSF	AddFP, DRCKCAP	InitSys, DCLS, DRCK, AddAutoPW	100	92.31	100	100

Table 1: Evaluation results for all 41 variability propagation runs from statemachine domain to component domain.

#	Source Component Delta	Initial Source Model	Target Statemachine Delta	Initial Target Model	RQ1.1	RQ2.1	RQ1.2	RQ2.2
1	InitSys		InitSM		100	100	100	100
2	AddCompEM	InitSys	AddRegEM	InitSM	100	5.83	100	10.00
3	AddHMI	InitSys	AddHMI_Core	InitSM	100	18.84	100	29.55
4	AddFP	InitSys	AddFP	InitSM	100	18.52	100	29.41
5	AddAutoPW	InitSys	AddAutoPW	InitSM	100	8.89	100	15.09
6	AddFPinAutoPW	InitSys, AddAutoPW	AddFPinAutoPW	InitSM, AddAutoPW	100	12.12	100	20.00
7	AddManPW	InitSys	AddManPW	InitSM	100	12.96	100	21.21
8	AddFPinManPW	InitSys, AddManPW	AddFPinManPW	InitSM, AddManPW	100	12.12	100	20.00
9	Mod_EM_Heating	InitSys, AddCompEM	Mod_RegEM_Heating	InitSM, AddRegEM	100	2.70	100	4.26
10	DLEDEM	InitSys, AddCompEM	DAddEMLEDEM, EMLedTop, EMLedLeft, EMLedBottom, EMLedRight	InitSM, AddRegEM	100	5.26	100	13.04
11	AddLED_Heatable	InitSys, AddCompEM, Mod_EM_Heating	AddLED_EMH	InitSM, AddRegEM, Mod_RegEM_Heating	100	11.11	100	20.00
12	DLEDFingerProtection	InitSys, AddFP	AddFP_LED	InitSM, AddFP	100	12.00	100	20.00
13	DLEDManPW	InitSys, AddManPW, AddHMI	AddManPW_LED, DAddHMILEDManPW	InitSM, AddHMI_Core	100	12.40	100	20.00
14	DLEDAPW	InitSys, AddAutoPW	AddAutoPW_LED	InitSM	100	10.00	100	16.13
15	DCLS	InitSys	AddCLS	InitSM	100	18.52	100	29.41
16	DCLS, AddWiringCLSManPW	InitSys, AddManPW	AddCLS, DAddManPWCLS	InitSM, AddManPW	100	5.00	100	9.62
17	DCLS, AddWiringCLSAutoPW	InitSys, AddAutoPW	AddCLS, DAddAutoPWCLS	InitSM, AddAutoPW	100	4.35	100	8.47
18	DAL	InitSys, DCLS	DAddCLSAL	InitSM, AddCLS	100	16.67	100	26.67
19	DRCK	InitSys, DCLS	AddRCK, DAddCLSRCK	InitSM, AddCLS	100	15.63	100	25.00
20	DLEDCLS	InitSys, DCLS	AddLEDCLS	InitSM	100	11.11	100	17.65
21	DRCKSF	InitSys, DCLS, DRCK	DAddRCKSF	InitSM, AddRCK	100	5.00	100	13.33
22	DAS	InitSys, AddHMI	AddAS, DAddHMIAS	InitSM, AddHMI_Core	100	14.36	100	23.33
23	DCLS, DRCK, DAS, DCAS	InitSys, AddHMI	AddCLS, AddRCK, AddAS, DAddHMIAS, DAddASCAS	InitSM, AddHMI_Core	100	13.92	100	22.68
24	DLEDAPWCLS	InitSys, AddAutoPW, DLEDAPW, DCLS	DAddLEDAutoPWCLS	InitSM, AddAutoPW_LED	100	4.21	100	7.14
25	DASIM	InitSys, AddHMI, DAS	DAddASIM	InitSM, AddAS	100	7.32	100	21.43
26	DLEDASIM	InitSys, AddHMI, DAS, DASIM	LedASIM	InitSM	100	11.11	100	17.65
27	DLEDAS	InitSys, AddHMI, DAS	LedASAD, LedASAL, LedASAC, DAddHMILEDAS	InitSM, AddHMI_Core	100	12.09	100	19.30
28	DRCKCAP	InitSys, DCLS, DRCK, AddAutoPW, AddFP	DAddRCKCAP, DAddRCKCAPSF	InitSM, AddAutoPW, AddRCK	100	3.23	100	5.88
29	AddFP, Wire_HMI_FP	InitSys, AddHMI	AddFP	InitSM, AddHMI_Core	100	14.81	100	25.00
30	AddManPW, Wire_HMI_ManPW	InitSys, AddHMI	AddManPW	InitSM, AddHMI_Core	100	9.26	100	16.13
31	AddAutoPW, Wire_HMI_AutoPW	InitSys, AddHMI	AddAutoPW	InitSM, AddHMI_Core	100	6.67	100	11.76
32	AddCompEM, Wire_HMI_EM	InitSys, AddHMI	AddRegEM	InitSM, AddHMI_Core	100	4.04	100	7.14
33	AddFP, Wire_FP_ManPW	InitSys, AddManPW	AddFP	InitSM, AddManPW	100	14.81	100	25.00
34	AddFP, Wire_FP_AutoPW	InitSys, AddAutoPW	AddFP	InitSM, AddAutoPW	100	14.81	100	25.00

Table 2: Evaluation results for all 34 variability propagation runs from component domain to statemachine domain.