

Reference Attribute Grammars as an Incremental Validation Framework for Delta-Based Multi-Model Product Lines

Andreas Domanowski
Technische Universität Dresden
Dresden, Germany
andreas.domanowski@tu-dresden.de

Karl Kegel
Technische Universität Dresden
Dresden, Germany
karl.kegel@tu-dresden.de

Uwe Aßmann
Technische Universität Dresden
Dresden, Germany
uwe.assmann@tu-dresden.de

Abstract

A software product line comprises modules of core functionality as well as variable, configurable features. In model-driven engineering, the core consists of one or more models, and the features are defined by model transformations. Although developing such a product line brings many benefits, it poses two major challenges: First, all involved meta-models, transformation tools, and validation functions must be consistent, i.e., compatible. Second, the consistent co-evolution of models and features, i.e., all induced model transformations, must be assured.

This work explores a novel approach for tackling both challenges. By using *Reference Attribute Grammars* (RAG), a modeling technique known from compiler construction, it becomes possible to express all involved models and their validation logic uniformly. During model evolution, RAGs enable efficient incremental consistency checking and repair. We demonstrate the feasibility of a RAG-based product line in a case study.

CCS Concepts

• **Software and its engineering** → **Software product lines; Model-driven software engineering.**

Keywords

Model-Driven Engineering, Modeling Languages, Consistency, Variability, Attribute Grammar

ACM Reference Format:

Andreas Domanowski, Karl Kegel, and Uwe Aßmann. 2026. Reference Attribute Grammars as an Incremental Validation Framework for Delta-Based Multi-Model Product Lines. 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, 5 pages. <https://doi.org/10.1145/3837062.3839068>

1 Introduction

Software product lines (SPL) describe highly configurable software systems comprising a core functionality that can be combined with a variable set of features [18]. For realizing a concrete software product, the features of choice are selected. A feature may depend on, require, or prohibit the existence of other features. These relationships are expressed via *feature models* [18]. For developing SPLs, *delta-oriented programming* [19] is an established technique.

In that case, features are realized as *deltas*, i.e., sequences of incremental transformations (*delta operations*). Looking at SPLs from a model-driven perspective, the core consists of a model or a set of interrelated models from heterogeneous domains. In the latter case, we call such a system a (*multi-model SPL*).

Problem Statement and Research Question: A multi-model SPL comprises a set of interrelated core models, a set of model transformations for expressing features, and a feature model that specifies which feature combinations form valid products. Within the scope of this work, we restrict the feature-defining model transformations to *delta models* [19]. This stack of different models makes the design and evolution of a multi-model SPL inherently complex. Particularly, we identify two major problems:

- (1) *Environment Consistency:* Meta-models, transformation tools, and validation functions are typically realized in heterogeneous formalisms (e.g., Ecore, ATL, OCL), requiring adapter layers that themselves must be maintained. This fragmentation means a change in one artifact can silently invalidate another with no automated cross-check.
- (2) *Model Consistency:* All involved models must be consistent, i.e., free of contradictions. While creating, evolving, or removing any of these models, developers need to be aware of inconsistencies or get notified in a timely manner.

Consider a developer who renames a core model element. A delta referencing the old name may pass validation undetected, because the validator and the transformation tool rely on separate formalisms with no shared representation, requiring ad-hoc checks. The derived product silently becomes malformed. This illustrates how both problems compound: tool chain fragmentation prevents automatic cross-checking, and the absence of a shared model representation means the error surfaces only at product derivation time or even not at all. We pose the research question: **Can a single unified formalism simultaneously eliminate toolchain fragmentation and provide incremental, continuous consistency checking across all artifacts of a multi-model SPL?**

Approach and Contribution: This work explores using *Reference Attribute Grammars* (RAG) [8] as a unified technical foundation for multi-model SPL development. To the best of our knowledge, no prior work has yet applied RAGs in this role. RAGs are a structural fit for the stated problems: their synthesized and inherited attributes map naturally onto intra- and cross-model consistency checks, while reference attributes enable cross-subtree dependencies without explicit wiring, directly addressing both consistency problems identified above. All artifacts developed in the course of this work are made openly available as supplementary material [3].



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/10.1145/3837062.3839068>

2 Modeling and Validating SPLs with RAGs

This section starts with an introduction to RAGs and their application in Model-Driven Engineering (MDE). We continue by outlining how RAGs can serve as the technical foundation for a multi-model SPL solving both initially stated consistency problems.

2.1 RAGs for Model-Driven Engineering

Reference Attribute Grammars are a technique originating from compiler construction. An Attribute Grammar is a context-free abstract grammar that adorns nodes in the parse tree with attributes computed by functions that describe their semantics. These attribute functions can also be used to validate complex properties of the parse tree. *Synthesized* attributes are computed from information from children of a node, while *inherited* attributes are computed from information residing in parent nodes, making information about its surrounding context available and propagating it down the tree [9].

RAGs [8] extend Attribute Grammars with *reference* attributes. This enables the definition of non-local dependencies and therefore complex analytical computations. In particular, a reference attribute on a node of a subtree can resolve to a node in a distant subtree representing a different model, simplifying cross-tree analysis. Furthermore, RAGs have been extended into *Relational RAGs* [14] supporting non-containment relations over unidirectional and bidirectional references. This enables the definition of graph structures inside the grammar, not only on the attribute level. As a consequence, a Relational RAG enables the definition of metamodels equivalent to those in the *Essential Meta-Object Facility* [1]: model instances remain containment trees, enriched with typed, named cross-references between arbitrary nodes, retaining the structure of modeling language instances. Therefore, RAGs find wide adoption in MDE, as they can naturally express the syntax of a modeling language [13, 15, 20]. Furthermore, this syntax definition can be decorated with all the functionality needed to validate and manipulate the models of the defined language as attributes.

To support the practical use of RAGs, metacompilation systems such as *JastAdd* [4] enable the generation of executable object-oriented programs from grammar specifications and their attributions. Simply speaking, *JastAdd* provides attributions both for expressing context-sensitive constraints (during parsing), as well as for expressing such constraints on a runtime model. During the generation process of a *JastAdd* RAG, means for efficient analysis and manipulation of the language's parse tree are included in the generated program. This enables efficient incremental re-computations of attributes after manipulations of the underlying tree based on caching and dynamic dependency tracking [24].

2.2 SPL Definition and Validation with RAGs

Metamodel Definition: We approach the modeling of delta-based multi-model SPL by first defining grammars for all core metamodels of the multi-model (i.e., the core domains, see Figure 1 below). For each domain, we define the grammar for a delta language, i.e., a transformation language, that supports add, modify, and remove operations. Listing 1 presents an example. It showcases a grammar serving as a metamodel for a component system, and

a further grammar that metamodels deltas operations for this domain, e.g., the addition and removal of components to and from the model, and the modification of properties of existing components. Although this delta language can be engineered manually, its automation is explored in related research [6, 23]. Furthermore, we define a grammar for feature models. This grammar enables us to specify which combinations of features are possible, i.e., which form valid products when applied together.

Listing 1: Two demonstrative grammars in extended JastAdd-style for a domain metamodel and a derived delta language.

```
// Nonterminals are mapped to Java classes
// Terminals (in angle brackets) are mapped to fields

// Model consists of multiple model parts
ComponentModel ::= Component* ;
// Terminal and Children
Component ::= <Name: String> InputPort* OutputPort* ;

// Abstract nonterminal maps to an abstract class
abstract Port ::= <ID: String> <Signal: String> ;
// InputPort and OutputPort inherit Port
InputPort: Port;
OutputPort: Port; // cf. Alternative in EBNF
// Unidirectional named relation
rel OutputPort.from -> InputPort;

Delta ::= DeltaOperation* ;
abstract DeltaOperation* ;
// Add or modify Component (remove omitted for brevity)
AddComponent: DeltaOperation ::= <Name: String> ;
ModifyComponent: DeltaOperation ::=
    <ComponentToModify: String> <NewName: String> ;
// Add relation (modify, delete omitted for brevity)
AddPortRelation: DeltaOperation ::=
    <FromId: String> <ToId: String> ;
```

Automated Analysis and Validation: Now, having each part of a multi-model SPL structurally defined as an RAG, we can define functions (i.e., attributes and how they are computed) for analysis and validation.

Computing attributions may require access to information from all involved models. We therefore unify the previously defined grammars under a single root rule on whose right-hand side their root nonterminals are mentioned (tying together all core models, delta models, and the feature model in Figure 1 below). This results in the situation that the entire SPL is a single unified model. The remainder of this section presents the attribute functions used to realize consistency validation in a multi-model SPL. As of now, attribute functions are defined manually.

First, when selecting features to derive a product, the system must check that the feature combination is valid according to the feature model (*feature-product-consistency*). The solution is to define an attribute function in the feature model that checks the configuration against itself and the involved models. Since all required information resides in the feature model subtree, this can be implemented with a synthesized attribute. Second, when a valid combination of features, and, therefore, a derived sequence of deltas is selected, the applicability of the product-deriving model transformation must be validated (*delta-consistency*). In other words, the system must verify that the preconditions for each delta to be executed hold. The solution is to define several attribute functions

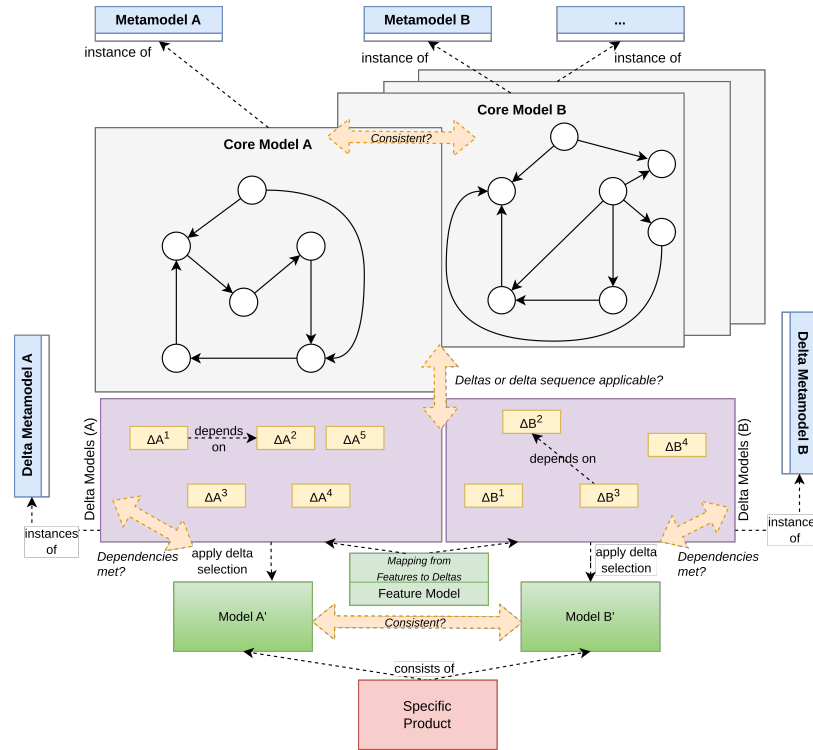


Figure 1: Overview of all models and metamodels in a delta-based multi-model SPL. Core models (white area) are instances of domain metamodels (blue, top). Deltas (purple area) are instances of delta metamodels (blue, sides) and may carry ordering *depends-on* constraints. A feature model (green) maps feature selections to delta sequences. Applying the selected deltas yields derived product models. *Consistent*-relations mark inter-model relations that must hold at all times between core models and between derived product models. Our approach unifies all these components as subtrees of a single RAG, so that every *Consistent*-relation is expressed as a reference-attribute computation between the corresponding subtrees.

in the delta grammar, with supporting attribute functions defined in the model grammars. Since information from different subtrees is required, a combination of inherited and synthesized attributes realizes this. Third, there may exist product-line-specific (also called domain-specific) constraints on the set of core models that must hold before and after the derivation of a product, i.e., the application of deltas. Constraints may exist within a core model (intra-model) or between models (inter-model). These constraints can be defined as attribute functions in the model grammars, both synthesized for intra-model validations, and inherited and synthesized for inter-model validations.

Addressing the Consistency Challenges: The presented modeling approach incorporates all structural and validation aspects of a multi-model SPL into a unified RAG. Although this choice requires a specific technical space, it solves both consistency problems by construction: First, the environment consistency problem is solved as a direct consequence of the RAGs’ ability to describe all aspects of the SPL in a single model. Second, the model consistency problem is solved by the validation functions defined in the RAG. An implementation in JastAdd realizes the attribute evaluation efficiently: Attribute values are only computed on demand upon access, and

dependencies to tree nodes or other attributes are tracked dynamically. When any of those changes, the cache of the depending attribute is invalidated. This realizes incremental evaluation and supports efficient SPL development. Simply speaking, whenever any kind of change happens to one of the involved models or model transformations (deltas) of the SPL, exactly those attributes are re-computed which are directly or indirectly impacted by the change. Consequently, validation, e.g., consistency assurance happens incrementally and continuously during modeling.

3 Case Study

We demonstrate the practicality of our approach by implementing a multi-model SPL, realizing an *Electric Door* product line loosely based on the domains of the Body Comfort System case study [11]. Besides modeling the domains and their variability (feature model, deltas) in JastAdd grammars, we implemented cross-model validation as attribution functions. Furthermore, our tooling supports the realization of products by applying deltas, i.e., transforming base models. For visual validation of the transformation, we implemented a PlantUML-based visualization of the resulting models. All models and software tools developed in this case study are available in the supplementary material [3].

Domain and Grammar Structure. Our exemplary product line comprises two core domains. A *state machine domain* models behavioral aspects: a *SystemStateDescription* groups multiple *StateMachines* with states and transitions carrying an optional trigger signal, guard condition, and action signal. A *component model* specifies structural aspects: a *ComponentModel* groups *Component* nodes, each offering typed *InputPort* and *OutputPort* children identified by signal names and connected via *ComponentConnection* relations. The case study comprises eight grammars: one per core domain, a delta language derived from each, a feature model grammar, a grammar for the *DeltaUniverse* (the collection of all available deltas), a *ModelUniverse* grammar uniting both core models, and a *UnifiedUniverse* root tying together the aforementioned grammars. This single root non-terminal makes the entire product line a single, coherent model for attribute evaluation.

Feature Model, Core Model, and Deltas. The feature model defines a root *Electric Door* feature with subgroups containing optional and alternative features (*Authentication*: PIN and alternative Face Recognition; *Security*: inclusive or of Lockdown, Intrusion Detection, and Auto-Close) Each feature has a corresponding set of deltas that contains transformations to the system state description and component model, e.g., adding or modifying states, transitions, components, and ports. Our base model defines a basic single component and its corresponding state machine, both named *DoorControl* with respective ports, triggers, and actions for basic control.

Consistency Checks. Three categories of consistency checks from Section 2 are implemented as attribute equations: *Feature-product-consistency*: A synthesized attribute on the *Unified Universe* collects all deltas mapped to the currently selected features and verifies that every delta's declared *dependsOn* set is fully covered. Selecting a feature whose companion delta depends on an unselected delta immediately yields *false*, without any external validation step.

We address Delta-consistency. Inherited and synthesized attributes in the delta grammar verify applicability of each delta operation against the current base model. A synthesized attribute on each state machine delta iterates its operations and delegates to a per-operation *isApplicable*-check: an *AddRegion* is inapplicable if a region with that name already exists; a *ModifyRegion* is inapplicable if the target region is absent or the proposed new name conflicts. The base model root is propagated into the delta subtree via an inherited attribute, realizing cross-subtree information flow without explicit wiring between otherwise independent grammar fragments.

Furthermore, we address inter-model consistency. Inherited and synthesized attributes propagate information across subtrees for cross-domain constraints. An attribute validates *Region-Component correspondence* the synthesized attribute by checking that every state machine has a corresponding component and vice versa. Per-region and per-component inherited attributes receive the opposite domain's list via equations on their common parent *ModelUniverse*, enabling name-lookup without hard-coding cross-tree dependencies. This check is implemented for both consistency between base models and realized products, i.e., base models that are transformed by delta sequences. Furthermore, another attribute validates *Transition-Signal correspondence*. Synthesized attributes on each *Transition* verify that the transition's trigger (resp. action) signal

name matches an input port (resp. output port) of some component. The *ModelUniverse* reference is inherited down to each *Transition* node, enabling it to query the component model without explicit cross-subtree coupling.

JastAdd's demand-driven evaluation with dynamic dependency tracking ensures that any model change triggers recomputation of exactly those attributes whose results depend on the changed node. Therefore, consistency feedback is incremental and continuous, enabling efficient development within one technical space.

Results and Feasibility. Automated test cases exercise the implementation. Product-derivation tests verify the correct number of regions and components after selecting features and combinations thereof. Further tests showcase the individual consistency attributes: delta applicability for *AddRegion* and *ModifyRegion* operations, region-component correspondence in the base model and in delta sequences, and transition-signal correspondence for triggers and actions. All checks pass without any external validation tooling, confirming that the three consistency categories are entirely expressible as RAG attribute equations within a single technical space and demonstrating both environment consistency (one formalism, one tool) and model consistency (continuous incremental checking). Since JastAdd's demand-driven evaluator tracks attribute dependencies dynamically, recomputation after a model change is bounded by the scope of the change rather than total model size: a localized edit revalidates only the transitively affected attributes, not the entire SPL. The supplementary material provides the full implementation [3].

4 Related Work

Different approaches for enabling the realization and validation of multi-model SPLs can be found in the literature. Greiner et al. [5] present how presence conditions and variability information are kept consistent across multiple related models in model-driven product lines. Vierhauser et al. [25] introduce consistency checks for heterogeneous models and artifacts in industrial product lines. Schröpfer et al. [21] propose checking consistency exclusively for realized products rather than reasoning about every possible one in advance. Lopez-Herrejon et al. [12] validate metamodel-conformance of artifacts in a model-driven SPL incrementally with the reuse of consistency checks. Close to our work, Ananieva et al. [2] propose an approach for managing variability with integrated consistency preservation atop the *Vitruvius* framework. In contrast to our unified formalism, their approach is deliberately non-invasive: it layers a separate variability metamodel over and delegates consistency restoration to an external tool. Most closely related, Neumann et al. [16] preserve consistency across domains in a delta-oriented product line, but realize it by composing multiple heterogeneous frameworks propagate variability via unidirectional consistency-restoring transformations. In contrast to both aforementioned approaches, our RAG-based approach instead unifies metamodels, deltas, feature models, and validation logic in a single formalism, addressing environment consistency by construction and providing continuous, incremental, bidirectional checking rather than repair triggered at the time of delta creation or application.

Furthermore, there is a body of literature on analysis of delta-based SPLs. Pietsch et al. [17] formalize delta modules as graph

transformations to analyze and refactor interrelations between deltas over a single model, whereas we address cross-model consistency in multi-model SPLs by uniformly representing models, transformations, and validation logic in one incrementally evaluated formalism. Haber et al. [7] study how delta-based software architecture product lines evolve, analyzing delta applicability and ordering constraints. Seidl et al. [22] investigate consistency checks between feature and delta mappings in context of the evolution of domain models and the said mappings. Kowal et al. [10] define levels of consistency in delta-oriented UML models and defines incremental checks based on these levels.

A common limitation of these approaches is that consistency management and variability modeling are treated as separate concerns that rely on external validation mechanisms. Existing work enables consistency checking either across models, for specific product instances, or with incremental evaluation. However, it typically lacks a unified representation that integrates models, transformations, and validation logic. This fragmentation complicates the co-evolution of multi-models and limits the reuse of analysis results. In contrast, our approach provides a single, uniform formalism and representation with inherent support for efficient, incremental evaluation. This unification enables consistency validation within one integrated framework.

5 Summary and Outlook

This work demonstrates that Reference Attribute Grammars (RAG) provide a unified and efficient foundation for modeling, validating, and evolving delta-based multi-model software product lines. By integrating models, transformations, feature models, and validation logic into a single formalism, the approach addresses both environment consistency and model consistency by construction. In particular, RAGs enable the specification of dynamic cross-model dependencies and support efficient incremental validation, allowing inconsistencies to be detected and resolved during system evolution. To summarize, we answer the posed research question affirmatively: RAGs as a single unified formalism can simultaneously eliminate tool chain fragmentation and provide incremental, continuous consistency checking across all artifacts of a multi-model SPL, as demonstrated by the feasibility study. In future work, we will enhance semantic analysis of RAG-based SPLs by further analyses and investigate the scalability of our approach with an industrial case study. Furthermore, we will transfer research from delta language generation to our approach to ease the modeling of deltas for any domains modeled with RAGs. Lastly, we will investigate the generation and reuse of generic validation attributions common to all delta-based SPLs, easing adoption of our approach.

Acknowledgments

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1608 – 501798263.

References

- [1] 2026. About the Meta Object Facility Specification Version 2.5.1. <https://www.omg.org/spec/MOF/2.5.1> [Online; accessed 22. Apr. 2026].
- [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* (Madrid, Spain) (VAMOS '18). Association for Computing Machinery, New York, NY, USA, 3–10. doi:10.1145/3168365.3168377
- [3] Andreas Domanowski. 2026. Reproduction Package for Reference Attribute Grammars as an Incremental Validation Framework for Delta-Based Multi-Model Product Lines. Online. doi:10.5281/zenodo.21154536
- [4] Torbjörn Ekman and Görel Hedin. 2007. The JastAdd system – modular extensible compiler construction. *Science of Computer Programming* 69, 1 (2007).
- [5] Sandra Greiner and Bernhard Westfechtel. 2021. On Preserving Variability Consistency in Multiple Models. In *Proceedings of the 15th International Working Conference on Variability Modelling of Software-Intensive Systems* (Krems, Austria) (VaMoS '21). Article 7.
- [6] Arne Haber, Katrin Hölldobler, Carsten Kolassa, Markus Look, Bernhard Rumpe, Klaus Müller, and Ina Schaefer. 2013. Engineering delta modeling languages. In *Proceedings of the 17th International Software Product Line Conference* (Tokyo, Japan) (SPLC '13).
- [7] Arne Haber, Holger Rendel, Bernhard Rumpe, and Ina Schaefer. 2012. Evolving Delta-Oriented Software Product Line Architectures. In *Large-Scale Complex IT Systems. Development, Operation and Management*.
- [8] Görel Hedin. 2000. Reference Attributed Grammars. *Informatica* 24, 3 (2000).
- [9] Donald Ervin Knuth. 1968. Semantics of context-free languages. *Mathematical systems theory* 2 (1968).
- [10] Matthias Kowal and Ina Schaefer. 2016. Incremental Consistency Checking in Delta-oriented UML-Models for Automation Systems. *Electronic Proceedings in Theoretical Computer Science* 206 (03 2016).
- [11] Sascha Lity, Lachmann, Malte Lochau, and Ina Schaefer. 2013. *Delta-Oriented Software Product Line Test Models - The Body Comfort System Case Study*. Technical Report.
- [12] Roberto Erick Lopez-Herrejon, Alexander Egyed, Salvador Trujillo, Josune De Sosa, and Maider Azanza. 2010. Using Incremental Consistency Management for Conformance Checking in Feature-Oriented Model-Driven Engineering. In *International Workshop on Variability Modelling of Software-Intensive Systems*.
- [13] Johannes Mey, Thomas Kühn, René Schöne, and Uwe Assmann. 2020. Reusing Static Analysis across Different Domain-Specific Languages using Reference Attribute Grammars. *The Art, Science, and Engineering of Programming* 4 (02 2020). doi:10.22152/programming-journal.org/2020/4/15
- [14] Johannes Mey, René Schöne, Görel Hedin, Emma Söderberg, Thomas Kühn, Niklas Fors, Jesper Öqvist, and Uwe Aßmann. 2020. Relational reference attribute grammars: Improving continuous model validation. *Journal of Computer Languages* 57 (2020).
- [15] Johannes Mey, René Schöne, Ariel Podlubne, and Uwe Aßmann. 2022. Specifying Reactive Robotic Applications With Reference Attribute Motion Grammars. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. doi:10.1109/ACSOS-C56246.2022.00035
- [16] 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 (SPLC-A '25)*. New York, NY, USA, 88–93.
- [17] Christopher Pietsch, Udo Kelter, Timo Kehler, and Christoph Seidl. 2019. Formal Foundations for Analyzing and Refactoring Delta-Oriented Model-Based Software Product Lines. In *Proceedings of the 23rd International Systems and Software Product Line Conference* (Paris, France).
- [18] Klaus Pohl, Günter Böckle, and Frank Van Der Linden. 2005. *Software product line engineering: foundations, principles, and techniques*. Vol. 1. Springer.
- [19] 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*, Jan Bosch and Jaejoon Lee (Eds.).
- [20] René Schöne, Johannes Mey, Sebastian Ebert, and Uwe Aßmann. 2020. Connecting conceptual models using relational reference attribute grammars. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. doi:10.1145/3417990.3421437
- [21] Johannes Schröpfer, Felix Schwägerl, and Bernhard Westfechtel. 2019. Consistency Control for Model Versions in Evolving Model-Driven Software Product Lines. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*.
- [22] Christoph Seidl, Florian Heidenreich, and Uwe Aßmann. 2012. Co-evolution of models and feature mapping in software product lines. In *Proceedings of the 16th International Software Product Line Conference - Volume 1 (Salvador, Brazil) (SPLC '12)*. Association for Computing Machinery, New York, NY, USA.
- [23] Christoph Seidl, Ina Schaefer, and Uwe Aßmann. 2014. DeltaEcore - A Model-Based Delta Language Generation Framework. In *Modellierung 2014, Wien*.
- [24] Emma Söderberg and Görel Hedin. 2012. *Incremental Evaluation of Reference Attribute Grammars using Dynamic Dependency Tracking*. Technical Report.
- [25] Michael Vierhauser, Paul Grünbacher, Wolfgang Heider, Gerald Holl, and Daniela Lettner. 2012. Applying a Consistency Checking Framework for Heterogeneous Models and Artifacts in Industrial Product Lines. In *Model Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg.