Continuous Management of Consistency and Inconsistencies in Multi-Model Development: A Conceptual Response to the ICMM 2026 Challenge*

Razieh Dehghani[1,*,†], Fabian Eger[1,†], Martin Armbruster[1,†], Muhammad Asim Minhas[1,†], Bowen Jiang[1,†], Benedikt Jutz[1,†], Anne Koziolek[1,†] and Ralf Reussner[1,†]

[1]KASTEL - Institute of Information Security and Dependability, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany

Abstract

In Model-Driven Development (MDD), managing consistency between models is a persistent challenge that is strongly influenced by stakeholder priorities, as maintaining consistency often involves trade-offs among competing concerns. This paper contributes to the (In-)Consistency Management in Modeling Challenge 2026 (https://doi.org/10.5281/zenodo.18599404) by discussing how the Vitruvius framework can support consistency preservation. We review the concepts provided by Vitruvius; particularly the Virtual Single Underlying Model (V-SUM) and Views. V-SUM enables multiple, potentially inconsistent internal models to be presented externally as a single coherent model, effectively hiding internal inconsistencies from users. Views further allow users to focus selectively on the elements that are relevant to specific tasks, rather than dealing with the full complexity of all model artifacts.

We explain how these concepts help address three key aspects of inconsistency management: 1) Time-related aspects, where inconsistencies arise due to the evolution of models over time; 2) Artifact-related aspects, which is due to the diversity of dimensions or concerns represented within different modeling artifacts; and 3) Skill-related aspects, which result from the varying perspectives, expertise, and responsibilities of different stakeholders and roles. Finally, we outline several open questions and research directions that should be explored in future work.

Keywords

Inconsistency Management, Model-Driven Development, Safety-Critical Systems, Vehicle Brake System Design, Consistency Preservation, View-Based Development

CEUR-WS.org/Vol-4338/ICMM-paper5.pdf

1. Introduction

When developing complex interdisciplinary systems like Cyber-Physical Systems (CPSs), including a Vehicle Braking System (VBS), particular challenges arise [1]. Within the context of Model-Driven Development (MDD), managing model consistency requires consideration of several aspects [2]. These aspects relate not only to the occurrence of inconsistencies across engineering artifacts but also to the mechanisms employed to manage them. Consistency itself depends on domain-specific objectives and therefore requires domain expertise, as it is closely linked to the semantic interpretation of models.

Considering the safety-critical nature of braking systems and their exposure to changing physical conditions, such as temperature variations or component wear over time, their design inherently involves multiple sources of complexity. These include the diversity resulting from the involvement

Joint Proceedings of the STAF 2026 Workshops: AgileMDE, GCM, ICMM, LLM4SE, TTC. Rennes, France, June 29–July 3, 2026.

*Corresponding author.

raziyeh.dehghani@kit.edu (R. Dehghani); fabian.eger@kit.edu (F. Eger); martin.armbruster@kit.edu (M. Armbruster); muhammad.minhas@kit.edu (M. A. Minhas); bowen.jiang@kit.edu (B. Jiang); benedikt.jutz@kit.edu (B. Jutz); koziolek@kit.edu (A. Koziolek); reussner@kit.edu (R. Reussner)

https://mcse.kastel.kit.edu/staff_raziyeh_dehghani.php (R. Dehghani); https://dsis.kastel.kit.edu/staff_fabian_eger.php (F. Eger); https://mcse.kastel.kit.edu/staff_martin_armbruster.php (M. Armbruster); https://dsis.kastel.kit.edu/staff_muhammad_minhas.php (M. A. Minhas); https://mcse.kastel.kit.edu/staff_bowen_jiang.php (B. Jiang); https://dsis.kastel.kit.edu/staff_benedikt_jutz.php (B. Jutz); https://mcse.kastel.kit.edu/staff_anne_koziolek.php (A. Koziolek); https://dsis.kastel.kit.edu/staff_ralf_reussner.php (R. Reussner)

0000-0003-4619-0914 (R. Dehghani); 0009-0009-6845-1071 (F. Eger); 0000-0002-2554-4501 (M. Armbruster); 0000-0002-1733-4539 (M. A. Minhas); 0009-0001-8168-0866 (B. Jiang); 0009-0006-2110-7159 (B. Jutz); 0000-0002-1593-3394 (A. Koziolek); 0000-0002-9308-6290 (R. Reussner)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-26

of stakeholders with different areas of expertise and artifacts originating from various engineering disciplines, as well as the heterogeneity arising from structurally and semantically different models and consistency rules. In addition, the continuous evolution of models through versions and design variants over the system lifecycle, together with development processes driven by iterative feedback and evaluation loops, further intensifies these challenges. The presence of non-determinism, due to context-dependent decisions that influence change propagation strategies, and the need for prioritization to balance trade-offs between competing design concerns, must also be addressed. Moreover, the criticality of explicitly managing safety-relevant properties and the extensibility required to incorporate new models along with corresponding consistency mechanisms contribute to the overall design complexity. This paper investigates how the aforementioned concerns can be addressed using the Vitruvius approach [3] in the context of the (In-)Consistency Management in Modeling Challenge 2026 [4]. In this setting, inconsistency is regarded as an "unavoidable phenomenon" that should be systematically managed rather than eliminated. The discussion is conducted at a conceptual level, making the insights potentially transferable to other consistency management approaches.

The remainder of this paper is structured as follows. section 2 introduces the Vitruvius concepts underlying the representation of consistency relations and inconsistencies. section 3 analyzes inconsistency management from product-, process-, and people-oriented perspectives, including safety, maturity, impact, and criticality considerations that influence the toleration of inconsistencies over time. section 4 demonstrates the application of these concepts within the challenge scenarios. Subsequently, section 5 discusses limitations, open research questions, and remaining challenges. Finally, section 6 summarizes the paper and outlines directions for future work.

2. Foundation

Klare et al. [3] introduced the Vitruvius framework [5], which enables the creation of purpose-specific view types on top of a Virtual Single Underlying Meta-Model (V-SUMM), depicted in Figure 1. In V-SUM (instance of V-SUMM), related models are kept consistent through the propagation of changes and execution of Consistency Preservation Rule (CPR), allowing the models to evolve as if they formed a unified system. The Domain-Specific Language (DSL) `Reactions` allows defining CPRs that specify how a change in one model should be propagated to another model to preserve the consistency. The CPRs are defined at the metamodel level to ensure that instances remain consistent during their execution. View types in combination with view generation transformations define and enable the creation of views, on which engineers can work in their respective domain, as a view contains domain-specific parts of the models, relevant for the engineer.

3. Review of Inconsistency Management Aspects

Figure 2 provides an overview of three high-level aspects of inconsistency management targeted in this challenge, together with examples illustrating how the V-SUMM concept can be applied to manage the associated concerns. In practice, a V-SUMM is not uni-dimensional; rather, the different aspects represented by separate V-SUMMs in the figure are inherently interrelated. This necessitates the notion of a V-SUMM of V-SUMMs, in which multiple V-SUMMs are maintained consistently as part of a unified structure. The figure therefore serves as a conceptual illustration of the different dimensions that may need to be covered, an integration that can be technically realized using Vitruvius. The perspective is also inspired by the elements defined in the SPEM (Software & Systems Process Engineering Metamodel) metamodel by OMG [6]. The Vitruvius approach is particularly relevant for CPSs such as VBSs, where consistency requirements are first established at an abstract design level and subsequently enforced across different model representations. For example, models created for the design of a a VBS using different modeling languages can be kept consistent by providing the corresponding metamodels and specifying the associated CPRs between them.

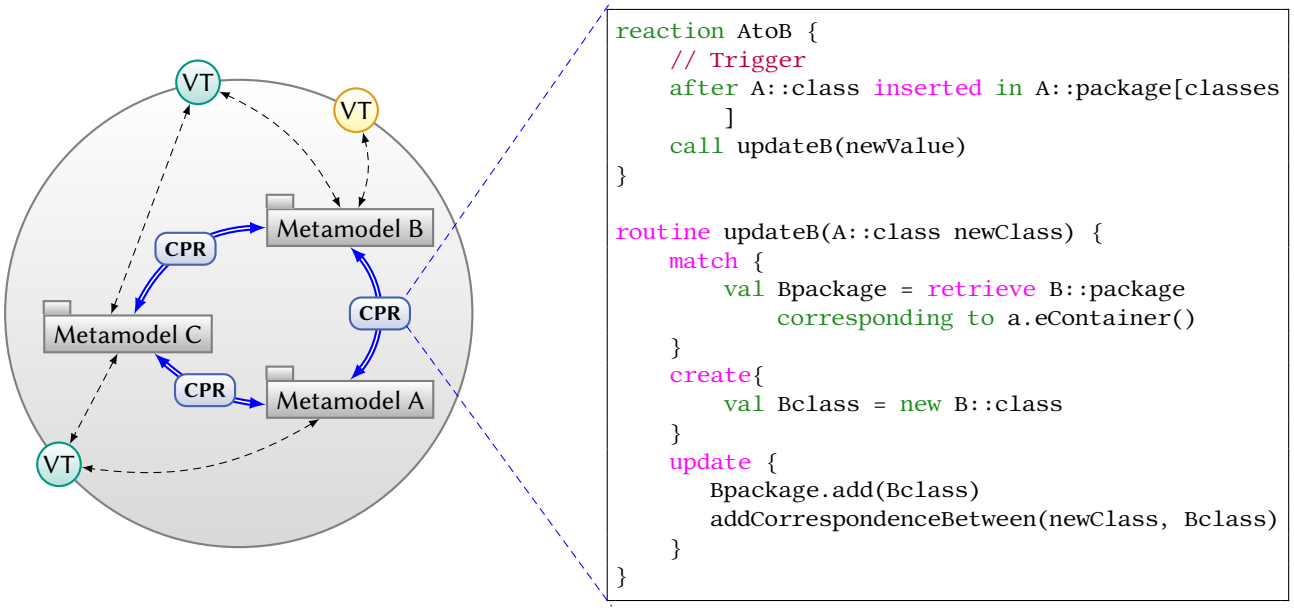


Figure 1: Visualization of a VSUMM (Meta-VSUM) based on [3]. Legend: MM = metamodels, VT = view types, ⟷ = consistency preservation rules (CPR), ←- -→ = view generation transformations.

The following subsections explain how VITRUVIUS can support preserving consistency despite the cross-cutting concerns, including safety and maturity. It should be noted that this section does not aim to address one scenario in isolation, but provides the reasoning framework for deciding whether an inconsistency is relevant, tolerable, safety-critical, stakeholder-specific, or maturity-dependent.

3.1. Supporting Consistency Preservation in the Presence of Cross-Cutting Concerns

The brake system challenge provides several interconnected models that follow an industrial RFLP workflow: a requirements model, a functional model, a structural model, a control and calibration model, and verification and testing results. The models (instances of metamodels in V-SUMM) describe some of the same parts of the system, which creates both useful opportunities and potential risks when dealing with cross-cutting concerns and inconsistencies. In the proposed approach, each type of artifact is treated as a view of a V-SUM rather than as a separate model with ad hoc trace links. The V-SUM incorporates only those RFLP-related artifacts that share semantics. These include braking functions, operating scenarios, components, parameters, and verification objectives. The V-SUM remains independent of any specific notion or tool. Various models of requirements, functions, structures, control, calibration, and verification are defined as projected views from the V-SUM. The synchronization of these views with the V-SUM is enabled through clear view mappings and consistency operators.

Support for Cross-Cutting Views All artifacts within the framework can be realized as a V-SUM-based view with explicit overlaps and crosscutting concerns.

- Requirements model view: This view is responsible for capturing safety, performance, and regulatory requirements as first-class citizens. These entities can be mapped further to functions, components, parameters and verification objectives in various V-SUMs. An important aspect is to support vertical (R→F→L→P) and horizontal (e.g., R→Safety, R→Verification) traceability.
- Functional Model View: It represents logical functions and behaviors of the brake system. Each function is linked to the corresponding V-SUM concept and to safety, performance and calibration concerns through orthogonal V-SUMs.
- Structural Model View: Describes physical decomposition, and its elements are identified with V-SUM entities. It also enables automated checks of functional allocation, safety mechanisms,

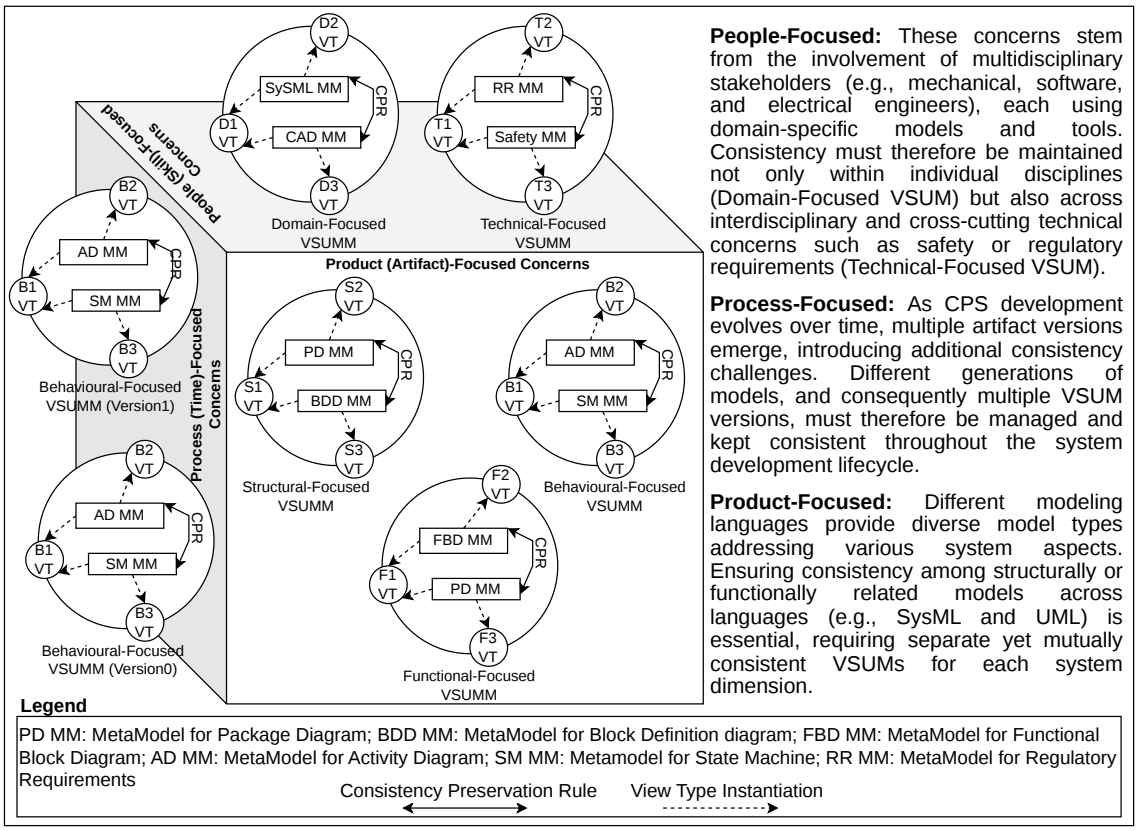


Figure 2: Use of V-SUMM for Managing Various Dimensions of Inconsistency, inspired by [3] and [6]

and hardware-software consistency.

- Control & calibration Model view: It encompasses all control algorithms, calibration data, control states, parameters, thresholds, while mapping to functional behaviors and hardware capabilities.
- Verification & test results view: Captures test specifications, results, coverage metrics projected from a verification V-SUM. This view explicitly relates verification artifacts to RFLP entities and to safety, performance, and calibration concerns. This enables the identification of inconsistencies between implementation and testing.

As all these views are projections from one or more V-SUMs with explicit consistency specifications, the semantic overlaps between artifacts are formally represented as correspondences rather than as implicit or tool-specific links. This is critical for systematically detecting inconsistencies and providing a controlled tolerance for inconsistency.

Orthogonal V-SUMs for Different Cross-Cutting Concerns Instead of a monolithic V-SUM, the proposed approach is based on utilizing orthogonal V-SUMs in order to handle distinct crosscutting concerns such as safety, performance, calibration and verification. Figure 1 represents a fundamental representation of one of such orthogonal V-SUMs and in practice there could be several orthogonal V-SUMs which means they intend to address orthogonal cross-cutting concerns yet they have semantic overlaps (Figure 2). The following V-SUMs can be applied:

- One V-SUM instance can be dedicated for "System Engineering concern" defined by RFLP artifacts. This will enable RFLP traceability between the brake system's functions and its hardware realizations [7].
- A second, orthogonal V-SUMinstance may be designated as "safety concern space", by integrating hazard analyses, safety requirements, safety mechanisms, safety test objectives, and relating them

to the RFLP-level elements through explicit inter-V-SUM mappings.
- Additional V-SUMs can be employed for calibration and control quality models (e.g., parameters, thresholds, operating conditions) and on verification evidence (e.g., testcases, simulation, and coverage metrics), again linked to the RFLP-oriented V-SUM.

In a nutshell, Orthogonal V-SUMs act as concern-specific semantic hubs that share selected elements or relations, providing specific views for safety, performance, and calibration to discuss the same brake torque function, sensor signals, or actuator capabilities, avoiding the need for a monolithic metamodel to represent concern-specific information.

3.2. Managing Safety and Maturity as Overarching Cross-Cutting Concerns

Managing Safety The challenge explicitly targets such approaches that allow working with inconsistencies, tolerating them over time rather than immediately removing them. The V-SUM approach addresses this issue by combining view-based modeling with explicit inconsistency representations that are part of the models.

- Representation & Explanation of Safety-related Inconsistencies: Inconsistencies are represented as first-class citizens in the V-SUM by correlating elements that conflict with other elements or reflect incomplete or outdated view projections. This is particularly relevant for safety-critical views such as safety requirements, functional safety mechanisms, and verification evidence [8]. Each inconsistency instance keeps track of the participating V-SUM elements (e.g., safety requirement, function to mitigate a hazard), the violated consistency rule, and some meta information such as detection time, origin and current status [9].
- Inconsistency Explanation through traceability: From an inconsistency, it is possible to navigate to associated hazards, system functions, components, and tests. This navigability provides a rationale of why the situation is inconsistent and how it can be interpreted as a relation to safety goals and hazards. This representation addresses the challenge's goal of making inconsistencies understandable and discussable within the engineering team.

Impact, Criticality & Safe Tolerance Over Time The impact of an inconsistency is evaluated by analyzing its reachability across the interconnected V-SUMs and views, which is particularly important for the safety-critical behavior of the brake system.

- The inconsistency impact metrics can be defined over the V-SUM (e.g., number of affected hazards, safety requirements, functions, components, and test cases). These metrics can be used to classify these inconsistencies along various dimensions such as safety-criticality, verification coverage, and development maturity.
- Safe tolerance is addressed through connecting the inconsistencies with "tolerance conditions" and engineering decisions. These conditions define under which assumptions an inconsistency may temporarily persist (for example, when there is a known verification gap that can be mitigated by operational restrictions or by using interim safety measures). This provision allows engineers to leave certain inconsistencies open during design iterations, while clearly documenting their safety impact and the conditions under which they are acceptable.

Managing Maturity A key challenge concerns understanding how maturity relates to the interaction between inconsistencies and overall system development progress. Maturity may be defined for each element involved in consistency preservation, including model elements and propagated changes. In this context, maturity reflects the extent to which an element is considered finalized from a domain expert's perspective. With respect to changes, considering maturity can help prevent the propagation of modifications that may later require rollback due to trial-and-error processes. It also reduces unnecessary disturbances to domain experts caused by the propagation of non-finalized changes. In terms of model elements, maturity awareness helps avoid building upon immature artifacts, which could otherwise

introduce additional inconsistencies and lead to further inconsistent system states in the future. Within Vitruvius, this can be supported by annotating both model elements and changes with predefined maturity levels.

4. Proposed Scenario-based Management Mechanisms for Inconsistency Management

The following subsections explain how specific aspects of inconsistencies, presented as challenge scenarios, can be addressed.

4.1. Handling Parameter Mismatches

A typical inconsistency in CPS engineering arises when a parameter is updated in one engineering model, while semantically related parameters still show the previous value. In our brake system scenario, this kind of inconsistency is present when the braking pressure limit is updated in the control and calibration model based on the latest test results, while the same limit in the functional model and the requirements model still holds. By using the `Reactions` language (Figure 1), the change in pressure limit in the control and calibration model will be propagated to the corresponding value in requireemnts model (Listing 1). This kind of inconsistency does not always have to be addressed as an immediate modeling fault. In collaborative and multi-view engineering, this kind of inconsistency might arise as a natural consequence of the modeling and design process [10, 11].

```
1 reaction ReqChange {
2  after attribute replaced at Control::Calibration[pressureLimit]
3  call changeReq(affectedEObject)}
4 routine changeReq(Control::Calibration controlcalibration) {
5  match {
6     val requirement_calibration = retrieve RequirementModel::CalibrationRequirement corresponding to
          controlcalibration}
7  update {requirement_calibration.pressureLimit = controlcalibration.pressureLimit}}
```

Listing 1: Example of Reaction and Routine Implementing Change Propagation

Representing Parameter Mismatches Explicitly Within a V-SUM, we propose representing parameter mismatches explicitly rather than treating them merely as temporary consistency violations. Building on Vitruvius 's explicit correspondences and CPRs [3], mismatches can be modeled as inconsistency elements linked to the affected model elements. These elements may be realized as separate inconsistency models or annotations on correspondence objects and store relevant information such as involved artifacts, conflicting values, origin, rationale, and resolution status. Technically, this requires extending existing CPRs so that conflicting changes may preserve consistency or create explicit inconsistency elements when immediate resolution is deferred. This turns mismatches into typed, queryable model elements and represents an explicit extension of Vitruvius.

Assessing Scope and Impact After explicitly representing a mismatch, the scope of a mismatch can be evaluated by tracing the established inter-model correspondences to directly affected artifacts. In a network of interrelated models, such dependencies might cut across multiple artifact boundaries. Accordingly, if adequate inter-model relations and consistency-preserving rules are established, impact analysis might be extended to derive a transitive impact set across the model network [12].

Informing Stakeholders Continuously Explicit inconsistency elements may also be used to facilitate stakeholder-specific communication. Vitruvius distinguishes between the role of the methodologist and the developer and derives views from the V-SUM via view types [3]. Based on this approach, role-specific views may be introduced to only display inconsistencies pertinent to specific stakeholder groups, such as control engineers, requirements engineers, or calibration engineers. The status changes

of an inconsistency element may be communicated via ordinary model evolution mechanisms, whereas the rationale and status history serve as the basis for long-term communication and decision-making.

4.2. Managing Uncertainty in Inconsistency Handling

If there are uncertain elements in a model, these uncertainties can be modeled within the V-SUM and annotated to the uncertain elements. To model an uncertainty, the unified uncertainty model by Mäkelburg et al. [13], for instance, can be utilized. Within this model, a modeled uncertainty includes its location (i.e., elements, where the uncertainty is observed), effect (i.e., its impact on the system, operations, outcomes, and decision-making processes), and further attributes. When the uncertainty causes an inconsistency, the inconsistency can be modeled as described in the previous subsection 4.1. Then, related inconsistencies and uncertainties can be linked implicitly over the uncertainty's location and inconsistency's involved elements or explicitly by introducing a correspondence between the uncertainty and inconsistency.

By linking an uncertainty and inconsistency, the attributes of the modeled uncertainty can be leveraged to support the inconsistency handling. On the one hand, this effect provides quantifiable categories [13] that enable the exploration of the uncertainty state space and its impact on (in)consistency, thereby supporting informed decision-making processes. On the other hand, attributes such as risk, temporal pattern, or kind allow deriving further properties of an inconsistency, for example, identifying if it is a critical or provisional inconsistency.

4.3. Managing Evolution-Relevant Aspects of Inconsistency

Three aspects of evolution targeted in the challenge include the severity of inconsistencies, their level of intolerance (i.e., strictness), and the need to make them visible and urgent. The persistence of changes, along with their associated metadata, such as severity and importance from a human perspective, can support the effective management of these issues. Furthermore, quantifiable metrics, such as the number of corresponding required changes across related models, can be captured using the correspondence model in Vitruvius to assess the impact of inconsistencies. In addition, maintaining separate branches for propagating changes in parallel enables the exploration and comparison of alternative resolutions with varying levels of strictness. Such information can be preserved across different branches representing various versions of the V-SUM. Conflicts arising during the merging of propagation branches may serve as indicators of unresolved inconsistencies.

Besides the use of separate branches to explore different design alternatives and separately develop features, the transactional editing of V-SUM (s) also aids in managing the evolution of inconsistencies. The two main paradigms for editing databases are Atomicity, Consistency, Isolation, Durability (ACID), which emphasizes maintaining consistency at all times, and Basically Available, Soft State, Eventually Consistent (BASE), which emphasizes the availability of databases [14]. ACID is strict with regards to enforcing the consistency of V-SUM s, as consistency-violating changes would be rejected. BASE would permit those changes. We envision the use of BASE in earlier stages of system development and for less mature parts of the V-SUM. For example, while the VBS requirements still need to be determined, the requirements model may still contradict the structural or functional models. Meanwhile, ACID should be used in later stages, and for more mature parts. Say that during production, a new ECU for controlling the brake should be installed to reduce cost, and thus the structural model is changed. CPSs propagate this change to the control model, where an inconsistency is detected: the new ECU does not have the computation performance for executing the control algorithms in time. Such a change should be rejected.

4.4. Managing Variants and Versions in Inconsistency Management

The standard Vitruvius framework (Section 2), needs to be extended to allow the handling of product line development and therefore variants and version management. The basic extension for the variability aspects is explained in [15]. Thus, the V-SUM is extended with an additional metamodel for the variability

modelling artifacts like a Feature Model (FM). In the approach by Eger et al. [15], the solution space artifacts are modeled through *Delta-Oriented Variability Modelling* [16, 17]. In order to synchronize the variability model and the solution space artifacts, additional Feature Consistency Preservation Rules (FCPRs) are defined, to have a traceability between the variability model and the solution space artifacts, as well as a mechanism to define rules how changes to the variability model affect the solution space and vice versa. Additionally to that, a metamodel is added representing the configuration of already derived products by representing the version of the product line the configuration belongs and which features of the FM are active in this configuration. Based on that preliminary setup the questions of the challenge can be addressed as follows.

Long-term inconsistent variants For the coexistence of inconsistent variants in the development process, we need to distinct two cases. In a first place if the inconsistency lays between the product line core and the variant, then the variants must be based on another version of the product line and thus the inconsistency can be there as long as the product should be updated to the new version of the product line. Or the variant changes the core part of the current product line version, which would be then propagated to all other variants using this version, which would then trigger all affected products that they either need to update their product or create a new version of the core based on that change and let all other variants depend on the old version. In a second place if the inconsistency is on a feature, which is not part of the core product line, first of all no other variant is affected on that, as the optional feature can also be present in different version. Therefore the different variants possibly can have different versions of a feature. In that case that there exist an instance level CPR between two variants for that feature, as explained in [15], than the inconsistency will be resolved via the CPR.

Traceability between deployed and evolving models In order to allow the traceability between deployed and evolving variants, the configuration model strongly gets into focus. In the configuration models all derived products are saved and therefore also for already deployed models. By this means, for changes in the product line or variants in the development process, the constraint checking via VitruvOCL, can then show if the deployed products are still valid in the current product line state. Furthermore the new deltas added for these changes are also propagated to each domain and the variability models, which means an update for the deployed models can be also calculated based on the new deltas. If the product line version between the current developed and the deployed variants differs, the product line version of the deployed model configuration can be changed, to determine if the configuration is still valid in the new product line version. In that case, the deltas between the old version and the new version of the product line can then possibly be calculated and based on that the deployed variant updated.

Risk-aware Inconsistency Management In order to make the inconsistency management risk aware, additional VitruvOCL (Vitruvius Object Constraint Language) constraints can be added to allow for these inconsistency by design but also to keep track which configuration or variants potentially have inconsistencies in the current state. So for example the constraints of a FM can be also expressed in VitruvOCL constraints on the configurations stored in the V-SUM. If we then want to lift a variant corresponding to an older product line core version to the newer version, then we can change the version in the configuration, which then leads to VitruvOCl constraint based on the FM of the newer version getting applied to the configuration and potentially can detect that the current configuration is not yet valid for the new product line version. As explained in the example, these constraints then just identify and report certain inconsistency but not try to repair them, which kind of represents a risk-aware mechanism to allow long-term inconsistencies especially with already deployed models.

Table 1
Open Questions Across People, Product, and Process Dimensions

	People (Skill)	**Product (Artifact)**	**Process (Time)**
People (Skill)	–	**What** kind of expert judgment override automated change propagation?	**When** should skilled stakeholders intervene in change propagation?
Product (Artifact)	**How** should artifact maturity influence **who** is responsible for resolving inconsistencies?	–	**Where** in the lifecycle should inconsistencies be tolerated temporarily?
Process (Time)	**How** can time-critical propagation avoid disrupting domain experts?	**Why** and **when** should propagation be postponed for evolving artifacts?	–

5. Discussion of Open Questions

Considering the dimensions of inconsistencies illustrated in Figure 2, various open questions may arise from combinations of these dimensions using 5W-H Aspects ([18]). Table 1 presents examples of such questions. Ongoing efforts focus on extending Vitruvius to address several open challenges: (i) enabling the evolution of CPRs over time, particularly regarding how inconsistency repair rules adapt as systems evolve; (ii) incorporating human feedback across different dimensions of propagated changes to enable continuous improvement; (iii) leveraging natural language to better assist domain experts in maintaining consistency; (iv) developing automated or semi-automated consistency repair mechanisms; (v) providing rollback-aware change propagation for deletions and other destructive changes, where restoring earlier states may require recovering removed elements; and (vi) improving the identification of semantic overlaps between artifacts and the specification of suitable CPRs, especially in settings where consistency relations are domain-specific, context-dependent, or subject to change.

6. Conclusion

In this paper, we have demonstrated how the Vitruvius framework can help address the challenges outlined by the ICMM Challenge 2026 [4]. This includes cross-cutting concerns related to the propagation of changes with varying levels of maturity, as well as the consideration of overarching concerns such as safety. Furthermore, we explained how the V-SUM concept and the `Reactions` language contribute to managing uncertainty in change propagation, supporting model evolution, and handling different model variants and versions. In general, we categorized inconsistency management concerns into three aspects, reflecting variations in stakeholder expertise, diversity of artifacts, and temporal constraints. Finally, we discussed several open questions arising from the interplay of these aspects and outlined future directions for extending Vitruvius, and explained the ongoing efforts to address the limitations.

Acknowledgments

This research is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1608 – 501798263. Benedikt Jutz is one of the contributors to the challenge description.

Declaration on Generative AI

The ideas and initial draft of this work were developed by us. GPT-5.3 was used only to revise parts of the text, and we reviewed the changes.

References

[1] R. Alur, Principles of cyber-physical systems, MIT press, 2015.

[2] W. Torres, M. G. Van den Brand, A. Serebrenik, A systematic literature review of cross-domain model consistency checking by model management tools, Software and Systems Modeling 20 (2021) 897–916.

[3] H. Klare, M. E. Kramer, M. Langhammer, D. Werle, E. Burger, R. Reussner, Enabling consistency in view-based system development—the vitruvius approach, Journal of Systems and Software 171 (2021) 110815.

[4] T. Weber, A. Lange, H. Klare, J. Cederbladh, T. A. Völk, K. Bause, B. Jutz, N. Hagel, M. Kaushik, C. Atkinson, Icmm ((in-)consistency management in modeling) challenge 2026 - construction of a brake, 2026. URL: https://doi.org/10.5281/zenodo.18599405. doi:10.5281/zenodo.18599405.

[5] Vitruv-Tools Contributors, Vitruvius: View-Based Development and Model Consistency Framework, https://github.com/vitruv-tools/Vitruv, 2025. Accessed: 2026-03-31.

[6] Object Management Group, Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0, Technical Report, Object Management Group, 2008. URL: https://www.omg.org/spec/SPEM/2.0/About-SPEM, accessed: 2026-03-31.

[7] T. Li, H. Lockett, C. Lawson, Using requirement-functional-logical-physical models to support early assembly process planning for complex aircraft systems integration, Journal of Manufacturing Systems 54 (2020) 242–257. doi:10.1016/j.jmsy.2020.01.001.

[8] S. Ananieva, Consistent management of variability in space and time, in: Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume B, SPLC '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 7–12. URL: https://doi.org/10.1145/3461002.3473067. doi:10.1145/3461002.3473067.

[9] J. Meier, Ensuring Inter-Model Consistency, Ph.D. thesis, Carl von Ossietzky Universität Oldenburg, 2024.

[10] I. Dávid, E. Syriani, C. Verbrugge, D. Buchs, D. Blouin, A. Cicchetti, K. Vanherpen, Towards inconsistency tolerance by quantification of semantic inconsistencies., in: COMMitMDE@ MoDELS, 2016, pp. 35–44.

[11] N. Weidmann, Tolerant consistency management in model-driven engineering, in: Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, 2018, pp. 192–197.

[12] P. Stevens, Maintaining consistency in networks of models: bidirectional transformations in the large, Software and Systems Modeling 19 (2020) 39–65.

[13] J. Mäkelburg, D. Perez-Palacin, R. Mirandola, M. Acosta, Surveying uncertainty representation: A unified model for cyber-physical systems, 2025. URL: https://arxiv.org/abs/2503.23892. arXiv:2503.23892.

[14] D. Pritchett, BASE: An Acid Alternative, Queue 6 (2008) 48–55. URL: https://doi.org/10.1145/1394127.1394128. doi:10.1145/1394127.1394128.

[15] F. Eger, L. König, T. Weber, D. Neumann, E. Burger, K. Feichtinger, R. Reussner, Towards consistent, variability-aware views in systems development, in: In VARIABILITY 2026 Proceedings, accepted for publication, 2026. Accepted for publication.

[16] I. Schaefer, L. Bettini, V. Bono, F. Damiani, N. Tanzarella, Delta-oriented programming of software product lines, in: J. Bosch, J. Lee (Eds.), Software Product Lines: Going Beyond, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 77–91.

[17] I. Schaefer, Variability modelling for model-driven development of software product lines, in: Fourth International Workshop on Variability Modelling of Software-Intensive Systems, volume 37 of *ICB-Research Report*, 2010, pp. 85–92. URL: http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf.

[18] H. D. Lasswell, The structure and function of communication in society, The communication of ideas 37 (1948) 136–139.