A Formalized Classification Schema for Model Consistency

Technical Report

Thomas Kühn[1], Dominik Fuchß[2], Sophie Corallo[2], Lars König[2], Erik Burger[2], Jan Keim[2], Manar Mazkatli[2], Timur Sağlam[2], Frederik Reiche[2], Anne Koziolek[2], and Ralf Reussner[2]

[1]Institute of Computer Science – Martin Luther University Halle-Wittenberg, Germany
[2]KASTEL – Institute of Information Security and Dependability, Karlsruhe Institute of Technology, Germany

17 July 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Abstract

Although *consistency* is a widely used term, there is no common understanding in the modeling community on what consistency precisely means for model-based engineering. Nevertheless, most engineers would agree that models should be consistent to a certain degree to be useful. Although there are use case specific definitions of consistency and its properties, none of them are generally and independently applicable. As a result, we observed that engineers rarely employ a precise definition for their consistency notion. In addition, most engineers consider that models are consistent, neglecting that consistency usually relates model elements instead. To remedy these issues, we established a focus group of domain experts to derive a generally applicable and fine-grained notion of consistency. Using this fine-grained notion of consistency, we introduce a formalized classification schema for consistency covering seven dimensions. Afterwards this classification schema was applied by each domain expert to one of seven widely different model-based engineering scenarios. In turn, our schema provides a precise model-element-based definition of consistency and vocabulary for the modeling community to distinguish consistency relations. Above all, we aim to raise awareness for different consistency notions and facilitate a common understanding of consistency in the MBE community.

1 Introduction

In model-based engineering (MBE), software and systems are described using multiple models, whereas each describes a system from a different viewpoint. Thus, no single model represents all information necessary to create, deploy, and maintain a complete system. Therefore, in many situations, models describing the system need to be *consistent* with each other. While most engineers would see this as a trivial statement, there is no common understanding of *consistency* in the MBE community (and in software engineering in general). As a result, it is hard to discuss and find commonalities or differences between consistency relations. To the best of our knowledge, the most general definition of consistency in MBE was introduced by [3], where consistency of two model elements is defined as membership in a binary relation. However, limiting consistency relations to binary relations neglects consistency relations spanning arbitrary many models, such as commonalities in [34]. Apart from a classification in terms of relation properties from set theory, e.g., uniqueness or totality, this consistency relation is not described any further. Besides [3], most other formal definitions in the literature, e.g., [68], [13], describe consistency in relation to consistency checking, consistency preservation, or consistency repair. Usually, the notions are tailored to their specific use case and domain and, thus, unsuitable for scenarios with multiple heterogeneous models. In most cases, however, the term is used informally. As a result, it becomes hard to distinguish, communicate and keep an overview of the various notions of consistency.

In this paper, we aim to provide a formal description of fine-grained n-ary consistency relations and formalize seven classifying properties of consistency relations that are independent of the specific use case. In particular, we will answer the following research questions:

(RQ1) *How can consistency relations be precisely defined?*

(RQ2) *What use case independent properties classify consistency relations?*

To this end, we performed an initial exploratory database search with Google Scholar for descriptions of consistency in (model-driven) software engineering to identify existing notions of consistency (section 2). With these notions as starting point, we established a focus group of domain experts to derive a generally applicable and fine-grained notion of consistency relations. As a result, we propose a novel classification scheme for consistency relations encompassing seven classifying properties focusing on consistency among model elements rather than models (section 3). We introduced our schema to the focus group and tasked each domain expert to apply it to an example in their domain. In sum, the experts described seven distinct MBE scenarios and successfully applied the classification scheme (section 4). To illustrate the applicability of the formalized classification schema, we additionally outline the analysis of one selected MBE scenario encompassing two consistency relations. Please note that we focused on consistency relations independent of the process they are used in, e.g., consistency checking, preservation, and repair. This entailed, that we omitted properties that are only tied to one of these processes.

2 Contemporary Notions of Consistency

2.1 Contemporary Consistency Relations

Consistency appears in various works and different contexts. While some authors share a common understanding, their terminology differs. This requires the clarification of what notions of consistency are used and what properties are associated with the term consistency. To the best of our knowledge, there is no common understanding of consistency, neither within a domain nor shared with others. Consequently, there are a variety of consistency notions in computer science literature that are unique and sometimes vaguely defined. Still, the notions are usually not generalized and thus not applicable for the MBE domain.

In contrast to most other domains, in mathematical logic and theoretical computer science, the term consistency is well defined [56]. Here, a consistent theory is a theory that does not entail a logical contradiction. This absence of contradictions is typically defined in terms of semantics. A theory is semantically consistent if there is a function (*interpretation*) that assigns a meaning to the symbols of a set of formulas under which all formulas of the theory are true (a theory has a *model*). However, in MBE, models are not always bound to a logical interpretation. In database systems, consistency is based on states of data and transactions between these states. In 1976, notions of consistency reached from *consistent* over *temporary inconsistency* (inconsistency between two transactions) to *conflict* [14]. Nowadays, terms like *strong consistency*, *weak consistency*, *eventual consistency*, and others are widely used [64]. All these terms are only defined for databases and not directly applicable to other domains. The same problem applies to other domains. Even though some notions transitioned from one domain to another, like *sequentially consistent* from multi-processors [38] to the caching domain [18], they are often not defined sufficiently general. In comparison to other domains, this notion also relates more to correctness than to other notions of consistency.

The distinction between consistency and completeness is also proposed by [24]. For them in the state-based requirements domain, completeness means that a response is specified for every possible input, whereas consistency refers to a specification being free of conflicting requirements and undesired non-determinism. Their understanding recurred some years later in another domain when [55] adopted it for dynamic migration of business processes.

Similar notions can also appear across different domains. [39] highlighted consistency as a security requirement for distributed systems with bilateral communication channels. In line with him, [75] consider equivalent criteria in machine learning but additionally distinguish between locally and globally applied criteria. Another example of cross-domain notions are *horizontal* and *vertical consistency* as well as *syntactic* and *semantic consistency*. Originally, [13] specify these terms for object-oriented behavioral models. Later, [27] adopted and expanded the terms for UML-based software development. However, they replace *horizontal* and *vertical consistency* with *intra-model* and *inter-model consistency*. We will show that these properties are not interchangeable. Although there seemed to be disagreement on consistency notions in UML-based software development, a later literature review from [44] on MBE publications

shows that the terms *horizontal*, *vertical* were meanly used according to their original definition. They also found uses of *semantic consistency*.

While previous notions assume binary consistency notions, [67] describes consistency for model transformations, such as QVT-R, as an n-ary relation. To handle these relations, she introduces association classes and treats them as regular binary consistency relations. Moreover, she introduces *inconsistent*, *shape-consistent*, and *perfectly consistent* as gradations for consistency. By contrast, [70] introduce *consistency conditions* that have to be preserved when the respective models change. Like [75], they distinguish between locally and globally applied conditions. Instead of classifying the criteria of consistency, [3] investigate properties of binary consistency relations during the synchronization of heterogeneous artifacts. They classify binary consistency relations as *bijections (one-to-one)*, *surjective functions (many-to-one)*, and *total relations (many-to-many)* depending on the mapping of the common property between two models.

While considering most of these earlier MBE approaches, [33] proposes various properties of consistency in the context of model transformations. Following [13] distinction between syntactic and semantic consistency, he defined *structural* and *behavioral consistency*. Like [68], he argues for the purpose of n-ary consistency relations. However, he claims that only structural n-ary consistency relations can be split into multiple binary relations because behavioral consistency relations cannot be easily decomposed in general. Besides that, he extends the scope of the properties of binary consistency relations [3] by distinguishing between *universally quantified*, as term for consistency relations that hold for all parts of a shared property, *existentially quantified*, as term for consistency relations that hold for at least some parts, and *statistical* as term for consistency relations that have a probability to be fulfilled.

In summary, we have revisited various consistency notions of different domains. We have highlighted that some established notions are very use case specific and bound to a special domain, e.g. notions from database systems. In some cases, e.g. multi-processors, consistency is even understood as a kind of correctness. However, there are also notions, like *intra-/inter consistency* and *syntactic/semantic consistency*, that recur in different domains. In MBE, we found consistency notions that are bijections, surjective functions, or total relations. They can be distinguished regarding local or global conditions and measured in gradations, quantifiable, or with statistical criteria. Other properties, such as structural and behavioral consistency are also considered relevant. While most consistency relations were described informally instead of explicitly formalized, the recurrence of many consistency notions across different domains hints at the generalizability of these notions. Finally, the changing terminology of consistency, missing references to existing consistency notions, and informal descriptions of many works reveal the need for a common understanding.

2.2 Contemporary Classification Schemes

Similar to us, [73] provide an overview of different consistency notions. Even though their paper is not yet peer-reviewed, they conducted a systematic literature review to provide an overview on consistency in context of model-driven engineering. They state that consistencies have a specification that defines how the consistency is expressed (constraint-based or transformational)

and an application scenario that it appears in, e.g., co-evolution. They proposed the *scope property* of consistency, which is divided into the categories: intra-model, inter-model, model-metamodel, and constraints. Moreover, they separate tolerance of these criteria from consistency as such. Furthermore, they identify properties of consistency relations, such as *relaxation*, *ranking*, *weighting*, *filtering*, and *temporal*. Most of these classes tolerate some inconsistencies. Based on their relaxation, ranking, weighting, and filtering, they define the quantifiability of consistency relations. Finally, they introduce the temporal category for approaches that tolerate inconsistencies until consistency is restored. We maintain that this is not a property of the consistency relation, but the process of consistency repair.

For business process transformations, [55] provide a taxonomy of different consistency notions. They treat business processes as petri nets and compare them. They derived four properties from which two are relevant consistency properties, in general. The first property defines how the trace sets are compared (*equal*, *subset*, *superset*). As discussed previously, this is part of the definition of the inconsistencies and therefore not a property of the consistency relation as such. The other property measures whether the consistency definition is based on the arrangement of the elements or their structure itself. However, we argue that this distinction is superficial, since the position of a model element in an arrangement can be encapsulated in the model element itself. Thereby, the consistency between two arrangements would be equal to a consistency of specific properties of model elements and, thus, a consistency based on structures. [61] provide a classification of consistency requirements of B2B integration approaches. They identify and locate different consistency relations by analyzing approaches within a general B2B integration scenario. Even though they create awareness for the different locations of consistency in the B2B integration domain, they do not derive general properties or requirements.

[57] derive a taxonomy of correctness criteria for database applications. Like us, they first distinguish between *consistency maintenance* and *consistency unit*. Consistency maintenance is beyond the scope of this paper, as we focus on consistency relations rather than the underlying process. In contrast, a consistency unit encompasses the data elements involved in a consistency, rather than the relation itself. They discern whether the entire database, a set of objects, or individual objects are the elements in the consistency relation. Still, this is also not relevant for the relation itself, as sets of objects can be considered as a single model element. In addition to this distinction, they derive correctness properties for their consistencies. Next to the database specific properties, they list the correctness of transaction results. This class is divided into absolute and relative categories, depending on whether the result of the transactions leads to a consistent state or the result is correct within a certain bound. This bound represents a tolerance criterion similar to [73].

In sum, we have collected contemporary classifications of consistency and noted that no publication, thus far, specifically targets consistency relations as such. None of them provided a formalized definition for consistency. While most taxonomies are limited to a specific domain, we found that similar properties appear in a wide range of research areas. Thus, we aim to create a common understanding and raise awareness of consistency relations and their properties across different domains.

3 Classification Schema

To derive our classification schema, we devised a four phase process. In the first phase, we aimed to identify commonalities and differences between consistency notions in different domains. Therefore, we conducted four bi-weekly focus group [36] sessions with five to eight domain experts from academia. Thereby, the experts were sensitized for different notions and we elucidated the different contexts, relations, prerequisites, and notions of consistency. In the second phase, we informally defined the consistency notions and asked the focus group to identify and describe consistency relations of representative MBE scenarios in their individual domains. In the third phase, we used the insights of all previous focus groups to formalize the consistency relations and classes. In the final phase, we introduced our formalization to the focus group and tasked the experts to apply it to the previously collected MBE scenarios. Henceforth, we introduce the formalized classification schema for consistency relations.

3.1 Mathematical Preliminaries

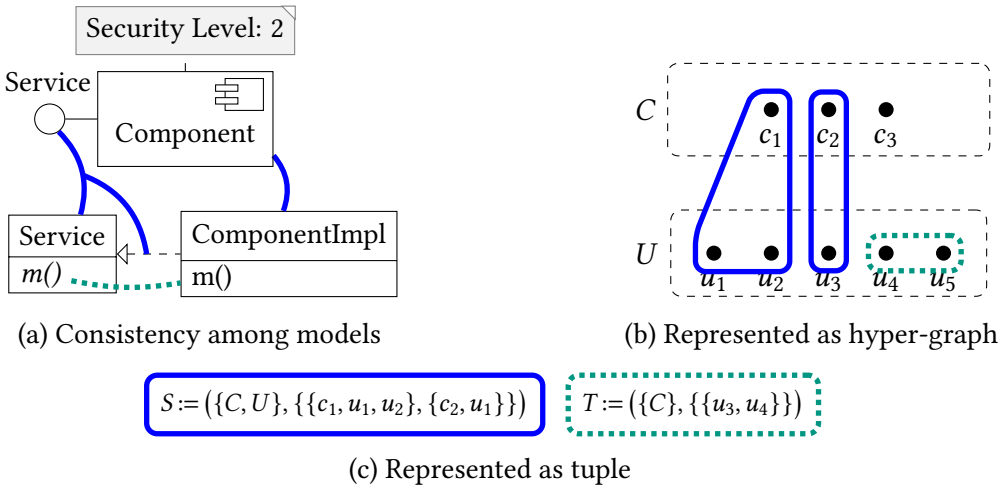
Notation Our formalization relies on first-order logic and set theory. We use the lower case letters i, j, and n to indicate natural numbers, whereas other lower case letters, e.g., m and k, denote model elements. The upper case letters R, S, and T denote consistency relations, whereas all other upper case letters, e.g., M, X, and Ω represent sets of model elements. All calligraphic letters, such as, $\mathcal{M}$, $\mathcal{E}$, and $\mathcal{P}$ denote sets of sets. Any (partial) (pre)orders are represented with binary operators, e.g., $\succ$, $\leq$, and $\triangleright$, and functions with lower case names, such as, e.g., lvl, $prop$, $phase$, and $score$. We write 2^M to denote the power set of set M, i.e., the set of all subsets of M.

Definition 1 *A* hyper-graph $G := (V, \mathcal{E})$ *consists of a set of vertices* V *and a set of hyper-edges* $\mathcal{E} \subseteq \{X \mid X \subseteq V \wedge |X| \geq 2\}$.

Please note that we require that each hyper-edge X in $\mathcal{E}$ connects at least two vertices from V and that the same hyper-edge X cannot occur multiple times in $\mathcal{E}$.

3.2 Formalization of Consistency Relations

[69] defines consistency relations as hyper-edges among models but not model elements. Thus, to provide a more fine-grained definition of consistency relations, we extend her definition to model elements. Hence, our consistency relation defines a hyper-graph over model elements of (different) models, whereas multiple hyper-edges exist that connect model elements with a joint purpose within the consistency relation.



(a) Consistency among models

(b) Represented as hyper-graph

(c) Represented as tuple

Figure 1: Illustrative consistency relations (thick lines) among a component model C and a corresponding UML class diagram U (a), their representation as hyper-graph (b) among model elements of C and U, as well as formal tuple (c).

Definition 2 *Let $\mathcal{M} := \{M_1, \ldots, M_n\}$ be a finite non-empty set of mutually disjoint models and $\Omega := \{m \in M \mid M \in \mathcal{M}\}$ the corresponding set of all model elements m in $\mathcal{M}$.*

Then a consistency relation $R := (\mathcal{M}, \mathcal{E})$ *consists of the set of models $\mathcal{M}$, as well as a set of correspondences $\mathcal{E} \subseteq \{X \mid X \subseteq \Omega \wedge |X| \geq 2\}$, whereas $X \in \mathcal{E}$ reflects a joint purpose of the included model elements, such that R defines a hyper-graph $G := (\Omega, \mathcal{E})$. R is denoted* non-empty, *iff $\mathcal{E}$ is not empty.*

Defined as such, consistency relations can span arbitrary many models, whereas each correspondence $X \in \mathcal{E}$ links at least two model elements. The hyper-graph defined by a consistency relation contains all model elements m of all models M that are part of the consistency relation. The hyper-edges of the hyper-graph directly reflect the correspondences connecting all model elements with a joint purpose w.r.t. the consistency relation. Considering the consistency relations illustrated in Figure 1a, the necessity for multiple hyper-edges per consistency relation becomes apparent. Here, the consistency of a component model with its realization in an UML class diagram is depicted (thick blue lines), whereas the Service and Interface (plus implements relation) correspond to each other as well as Component and ComponentImpl. While both correspondences belong to the same consistency relation, each is represented as a separate hyper-edge (see Figure 1b) to reflect the joint purpose of the corresponding model elements. Obviously, this does not exclude consistency relations with only one hyper-edge, e.g., the consistency between the interface method m() and its implementation in ComponentImpl (thick dashed green line). The corresponding formal notation is showcased in Figure 1c.

Employing this definition, we can precisely define the classifying properties of consistency relations. Henceforth, we will use the examples in Figure 1 to illustrate each property. Moreover, we provide additional examples and counter-examples for each classifying property.

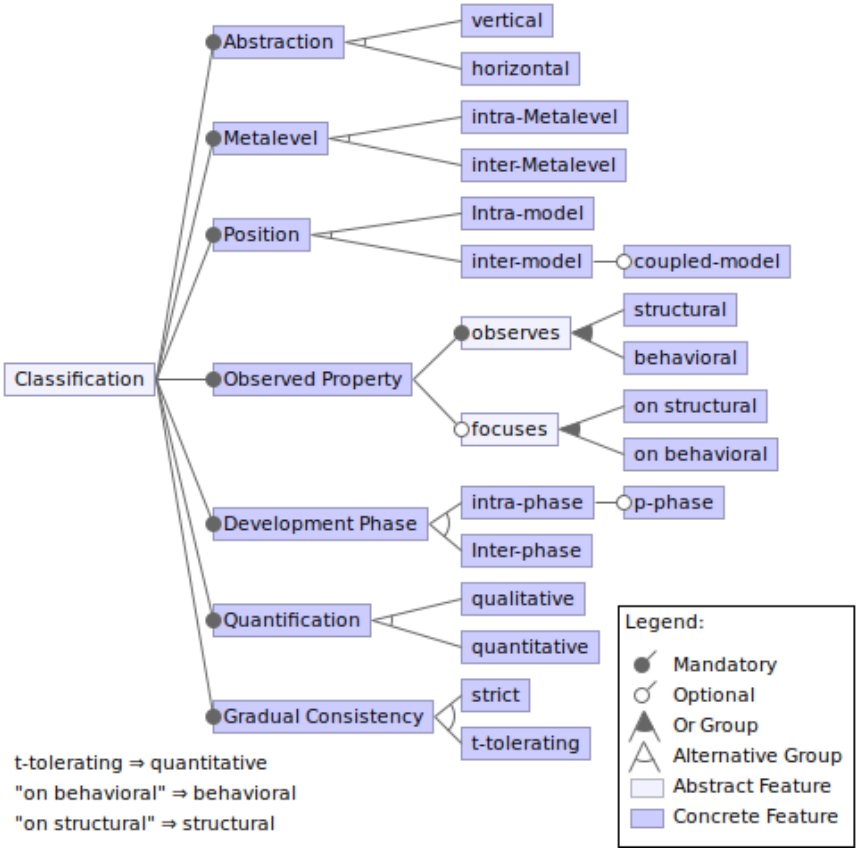


Figure 2: Feature model depicting our classification schema.

3.3 Classification Schema for Consistency Relations

As an overview for the following classification schema, in Figure 2, we present a feature model that concisely and comprehensibly captures the dimensions and classes of our schema. This figure, is meant to be employed in future systematic literature reviews or mapping studies.

3.3.1 Abstraction

According to [65], models are *abstractions* ("Verkürzungen", literally *reductions*) of their originals. Thus, they can be ordered by their level of abstraction. For example, specifications or requirements of a system are typically seen on a higher level, i.e., very abstract, while program code is seen as on a lower level, i.e., less abstract. The Model-Driven Architecture [50] defines models on three succinct levels of abstraction, i.e., the computation-independent model, platform-independent model, and the platform-specific model. Although one might assume levels to be strictly ordered, we argue that this is not always the case, especially, when models from different domains are involved. We acknowledge that some models, such as UML, encompass model elements of different levels of abstraction. In turn, we assign abstraction level to model elements. Like [13], we distinguish consistency relations that cross different levels of

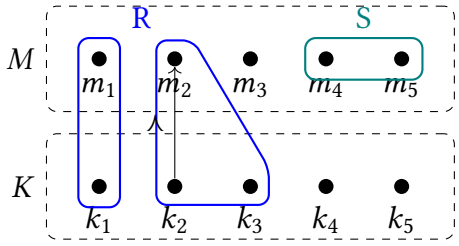


Figure 3: Illustration of horizontal (*S*) and vertical (*R*) consistency relations. The arrow from k_2 to m_2 indicates that m_2 is an abstraction of k_2.

abstraction from those that remain on the same level as *vertical* and *horizontal* consistency, respectively.

Definition 3 *Given a non-empty consistency relation R and a strict partial order $(\Omega, \succ)$, where for all model elements $m_1, m_2 \in \Omega$ with $m_1 \neq m_2$, $m_1 \succ m_2$ denotes that m_2 is an abstraction of m_1.*

R has vertical abstraction*, iff there is an $X \in \mathcal{E}$ with $m_1, m_2 \in X$, such that $m_1 \succ m_2$.*

R has horizontal abstraction*, iff there is no $X \in \mathcal{E}$ and $m_1, m_2 \in X$, such that $m_1 \succ m_2$.*

In short, a consistency relation is denoted *vertical*, if a correspondence (hyper-edge) connects model elements at different levels of abstraction. Considering the illustrative example (Figure 1), both Service c_1 and Component c_2 are more abstract then the corresponding Service u_1 interface and ComponentImpl u_3. As a result, given $c_1 \succ u_1$ or $c_2 \succ u_3$, the consistency relation S is vertical. By contrast, a consistency relation is *horizontal* if all model elements in all correspondences are on the same abstraction level. Conversely, T would be denoted horizontal.

In addition, Figure 3 illustrates the notion of horizontal and vertical abstraction using Def. 3. Here, we assume that only $k_2 \succ m_2$ holds, i.e., m_2 is an abstraction of k_2. This relation is shown as arrow from k_2 to m_2. Considering that $m_2 \succ k_2$ is the only abstraction, then *R* has *vertical abstraction*, as it contains a hyper-edge connecting the model elements m_2 and k_2. Conversely, *S* has *horizontal abstraction*.

3.3.2 Metalevel

According to [5], model elements can always be assigned to (exactly) one metalevel. These levels can be fixed, as in the classical four-level architecture of UML, or variable, as in multi-level/deep modeling approaches [5].

Definition 4 *Given a non-empty consistency relation R and a function $lvl : \Omega \rightarrow \mathbb{N}_0$ that assigns the metalevel to every model element $m \in \Omega$.*

R is intra-metalevel*, iff there is an $l \in \mathbb{N}_0$, such that for all $X \in \mathcal{E}$ and all $m \in X$ it holds that $lvl(m) = l$.*

R is inter-metalevel*, iff there is an $X \in \mathcal{E}$ and $m_1, m_2 \in X$, such that $lvl(m_1) \neq lvl(m_2)$ holds.*

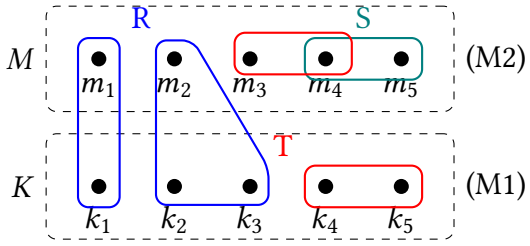


Figure 4: Showcase of intra-level (R) and inter-level (S) consistency relation, as well as one (T) that is neither. All elements of M belong to metalevel 2 and all of K belong to 1.

We distinguish intra-metalevel and inter-metalevel consistency relations by whether they connect model elements on the same or at different metalevels. While [27] considers the metalevel property to coincide with abstraction and refinement, we maintain that it as a special case of refinement (see subsubsection 3.3.1), where most horizontal relation are intra-metalevel, and inversely, most inter-metalevel relation are vertical. In fact, metalevels are coarse-grained in the dimension of abstraction and refinement, such that one might find an intra-metalevel consistency relation along a vertical abstraction, if elements of different abstraction levels are assigned to the same metalevel. This case is shown in Figure 1, where UML components and UML classes are connected. To showcase Def. 4 in Figure 4, we assign all model elements $k \in K$ to metalevel $lvl(k) = 1$ and all model elements $m \in M$ to metalevel $lvl(m) = 2$ Here, the consistency relation R is an *inter-level* consistency relation, whereas S is an *intra-level* consistency relation. Notably though, T is neither, as each hyper-edge connects elements of a different metalevel.

Although the model elements are on different levels of abstraction (cf. subsubsection 3.3.1), thus indicating a vertical consistency relation, they are on the same metalevel. Therefore, the consistency relation is also intra-metalevel. When specifying inter-metalevel consistency relations, one should additionally mention the crossed levels, e.g., *inter-metalevel between user model and metamodel.*

3.3.3 Position

Consistency can be defined within or between models. Of course, this consideration depends on the definition of what a *model* is, and where the model boundaries are. In theory, it is therefore often not relevant whether a consistency relation is inter- or intra-model [73]. In practice, however, this has many implications, since the model boundary also determines which tools are used, how models are persisted, and other aspects of the development process.

Definition 5 *Given a non-empty consistency relation R.*

R represents an intra-model *consistency, iff there is an* $M \in \mathcal{M}$ *such that for each* $X \in \mathcal{E}$ *it holds that* $X \subseteq M$.

R represents an inter-model *consistency, iff there is an* $X \in \mathcal{E}$ *and* $M_1, M_2 \in \mathcal{M}$ *with* $M_1 \neq M_2$, *such that* $X \cap M_1 \neq \emptyset$ *and* $X \cap M_2 \neq \emptyset$.

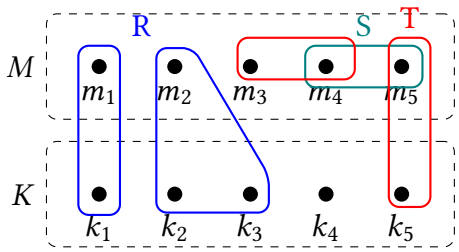


Figure 5: Depiction of intra-model (S), inter-model (T), and coupled-model (R) consistency relations.

R represents a coupled-model consistency, *iff* $|\mathcal{M}| \geq 2$ *and for every* $X \in \mathcal{E}$ *and every* $M \in \mathcal{M}$ *it holds that* $X \cap M \neq \emptyset$.

We say that a consistency relation is intra-model if it only connects model elements that are within the same model and inter-model if it connects model elements from different models. A coupled-model consistency relation describes an inter-model consistency relation where each correspondence between model elements contain elements of all involved models. All cases are depicted in Figure 1. The model C contains the elements c_1 and c_2, whereas U contains u_1, u_2, and u_3. The consistency relation S (thick blue line) between them is therefore an inter-model and, in particular, a coupled-model consistency. As the consistency relation T (thick green dashed line) only connects model elements of U, T is intra-model. Additionally in Figure 5, the distinction between intra-model and inter-model consistency relations becomes obvious, as it denotes whether hyper-edges cross the boundary of a model. In this case, R is a *coupled-model* consistency relation and S an *intra-model* consistency relation. In contrast to R, T is an *inter-model* consistency relation, yet not a *coupled-model* consistency relation.

Since model boundaries are often set such that elements in one model are at one level of abstraction, intra-model consistency is often horizontal.

3.3.4 Observed Property

As model elements describe the structure and/or behavior of a system, consistency relations can be distinguished between whether they observe structural or behavioral properties (represented by model elements). [33] argued that structural properties can be statically checked, such as the equality of values or the presence of elements, behavioral properties require dynamic analysis, e.g., abstract interpretation, model checking or co-simulation. Please note, that the purpose of a model determines which of these properties are relevant for its model elements.

Definition 6 *Given a non-empty consistency relation R, the set of observed properties $O := \{structural, behavioral\}$ and a labeling function $prop_\Omega : \Omega \rightarrow (2^O \setminus \emptyset)$ that assigns observed properties to all model elements $m \in \Omega$.*

R observes *a property $o \in O$, iff there is an $X \in \mathcal{E}$ and an $m \in X$, such that $o \in prop_\Omega(m)$ holds.*

R focuses on *a property $o \in O$, iff for all $X \in \mathcal{E}$ and all $m \in X$ $o \in prop_\Omega(m)$ holds.*

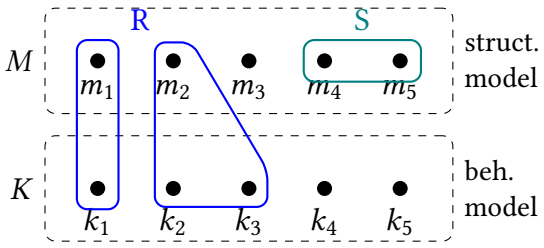


Figure 6: Here, all elements of model M observe structural properties, whereas elements of K observe behavioral features. Thus, while a consistency relation can focus on structural properties (S), others (R) observe both.

The function $prop_\Omega$ assigns observed properties, i.e., *structural* and *behavioral*, to each model element, whereas an element can observe both properties. A consistency relation is then said to *observe* one of these properties, if it contains at least one correspondence with a connected model element that observes this property. As this is a rather weak classification, we state that a consistency relation *focuses* on a property, if all model elements in all correspondences observe this property, albeit not exclusively. In the running example, all elements of the component model C and UML class diagram U observe a structural property of the system. However, the model element u_5 also observes the systems behavior, as it represents the implementation of method m(). Thus, both consistency relations S (thick blue line) and T (thick green dashed line) focus on structural properties, whereas only T also observes behavioral properties (via u_5). For the sake of simplicity, in Figure 6, we assume that all model elements $m \in M$ observe structural properties whereas all model elements $k \in K$ observe behavioral properties. Thus, when considering Def. 6, the consistency relation R observes both *structural* and *behavioral properties*. In contrast, the consistency relation S focuses on *structural* properties.

Besides that, our definition permits to adapt the set of observed properties to include other properties, e.g., *quality*.

3.3.5 Development Phases

Consistency can be expressed for model elements that *belong to* one or several phases in the life cycle of a system, e.g., specification time, design time, and run time. This notion should be understood as a model element being a main artifact of these development phases. The mere presence of a model element in a phase is not sufficient, since model elements of earlier phases are always relevant in later development phases, even if they are not directly used. Consequently, we distinguish between *intra-phase* and *intra-phase* consistency relations by whether they connecting model elements belonging to one phase or belonging to different phases.

Definition 7 *Given a non-empty consistency relation R, a non-empty set of development phases P and a labeling function $phase_\Omega : \Omega \rightarrow (2^P \setminus \emptyset)$ that assigns to each model element $m \in \Omega$ the development phases to which it belongs.*

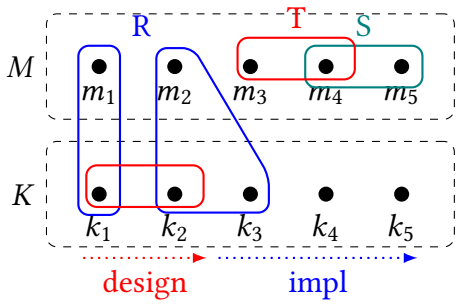


Figure 7: To illustrate implementation-phased (S), intra-phased (T) and inter-phase (R) consistency relations two arrows indicate, which model elements belong to the design and implementation phase, respectively.

R is intra-phase*, iff there is a $p \in P$, such that $p \in phase_\Omega(m)$ for all $X \in \mathcal{E}$ and all $m \in X$.*

R is inter-phase*, iff $\bigcap_{m \in X} phase_\Omega(m) = \emptyset$ for an $X \in \mathcal{E}$.*

In short, a consistency relation is denoted *intra-phase*, if all model elements in all correspondences belong to at least a given phase p. An *inter-phase* consistency relation contains a correspondence where there is no phase every connected model element belongs to. In the running example, we can assume that model elements of the component model belong to the specification time, whereas UML classes belong to the design time. Then, the consistency relation S is an inter-phase relation, whereas T is an intra-phase consistency relation. To illustrate *intra-phase* and *inter-phase* consistency relations introduced in Def. 7, we consider two development phases $P := \{design, impl\}$ in Figure 7. Lets assume $phase_\mathcal{M}(m_i) = phase_\mathcal{M}(k_i) = \{design\}$ if $i \in \{1, 2\}$ and $phase_\mathcal{M}(m_i) = phase_\mathcal{M}(k_i) = \{impl\}$ if $i \in \{3, 4, 5\}$. Then we can consider three different consistency relations. As a result, the consistency relation R is an *inter-phase* consistency relation, as for the hyper-edge $\{m_2, k_2, k_3\}$ the elements have no common phase, i.e., $phase_\mathcal{M}(m_2) \cap phase_\mathcal{M}(k_2) \cap phase_\mathcal{M}(k_3) = \emptyset$. Conversely, S is an *impl-phase* and consequently also an *intra-phase* consistency relation. In contrast, T is considered *intra-phased* and not *inter-phased*, as for each hyper-edge a unique phase is present albeit not the same between the hyper-edges.

While we do not assume a particular set of phases, we argue that, for a linear development process, this classifier is correlated with the level of abstraction since models of a later development phase refine models from earlier phases. In this case, all inter-phase consistency relations would also be vertical. In our opinion, this was Engel's understanding of horizontal and vertical consistency in [13], where vertical consistency is between elements before and after refinement, and horizontal consistency between elements at the same time of development. We maintain that these properties are indeed orthogonal, especially when considering non-linear development.

3.3.6 Quantification

While most consider consistency as a qualitative property denoting that corresponding model elements are either consistent or not, there are cases where consistency can be quantified, i.e., the connected model elements can be seen as more or less consistent [3]. This is useful for repairing inconsistencies, if there are multiple options improving consistency. Then, alternatives can be ordered by the degree of consistency they achieve, and the best one can be picked.

Definition 8 *Given a non-empty consistency relation R and a total function $score_\Omega : (2^\Omega \setminus \emptyset) \rightarrow [0,1]$ that assigns a* quantitative *measure to each possible hyper-edge.*

The function $score_\Omega$ is a consistency score *for the consistency relation R, iff $X \in \mathcal{E}$ implies $score_\Omega(X) > 0$ for all $X \in 2^\Omega \setminus \emptyset$. Then R is denoted* quantifiable *with $score_\Omega$.*

The $score_\Omega$ functions assigns a value to each potential correspondence (hyper-edge), indicating how consistent the connected model elements are. This function is a consistency score for a consistency relation, if all correspondences yield a value greater then zero. We then say that the consistency relation is quantifiable with this scoring function. We consider that lower values indicate less consistency among connected model elements, whereas higher values indicate a higher degree of consistency. Thus, a correspondence with a score of 0 means that the encompassed model elements are inconsistent. More importantly, we require that $score_\Omega$ is normalized to the interval $[0,1]$, while we concede this to be a sizable restriction, we maintain that in most cases a given scoring function can be normalized. For the running example, we could employ a normalized string similarity measure, e.g., a normalized Levenshtein distance, to compare the names of model elements. Then, the consistency relation S is quantifiable with this score. The correspondence between c_2 and u_3 yields ≈ 0.69, as c_1 is named Component and u_1 ComponentImpl. In contrast, the correspondence between c_1, u_1 and u_2 would yield 1.0, as the named model elements (c_1 and u_1) are named Service. To illustrate quantifiable consistency relations, we need to introduce a simple scoring function over $\Omega := M \cup K$:

$$score_\Omega(X) = \frac{2\,|\{i \in \mathbb{N} \mid m_i, k_i \in X\}|}{|X|}$$

Simply put, this scoring function determines the fraction of pairs $m_i, k_i \in X$ with the same index $i \in \{1, \ldots, 5\}$ in X, whereas $m_i \in M$ and $k_i \in K$. Albeit not useful in practice, this function is helpful to illustrate the quantification of consistency relations shown in Figure 8. Here, the score of each hyper-edge is shown close to the center of each rounded box in the color of the corresponding consistency relation. Consequently, we can see that both consistency relations R and S are *quantifiable* with the *score* function. In contrast, T is *not quantifiable* with the scoring function, due to the score of 0.0 of its hyper-edge $\{m4, m5\}$. Although *score* cannot quantify T, there might be scoring functions that make T quantifiable.

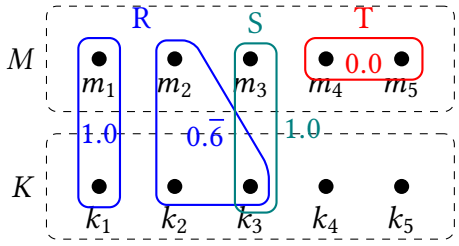


Figure 8: Showcase of consistency relations quantifiable (*R* and *S*) and not quantifiable (*T*) by the scoring function *score*. Here, each hyper-edge is annotated with its score.

3.3.7 Gradual Consistency

There can be criteria that lead to the gradual tolerance of inconsistencies between models [73]. This tolerance is usually tied to a quantifiable notion of consistency; if the system reaches a certain threshold in this quantified notion, it is declared consistent. To put it bluntly, although the model elements are not fully consistent, they are considered *consistent enough*. This acceptance can also be determined by further composition of other consistency notions, e.g., if a sufficient number of sub-systems are consistent or by defining a path to a consistent state that can be reached eventually. We distinguish between *tolerating* and *strict* quantifiable consistency relations.

Definition 9 *A consistency relation R is* t-tolerating *with* $0 < t < 1$, *iff it is quantifiable with* $score_\Omega(X)$ *and* $score_\Omega(X) \geq t$ *for all* $X \in \mathcal{E}$. *R is* strict, *iff it is qualitative or it is quantifiable with* $score_\Omega(X) = 1$ *for all* $X \in \mathcal{E}$.

A t-tolerating consistency relation reflects that a quantifiable consistency relation considers those correspondences as consistent, whose $score_\Omega$ yields a value greater or equal then t. Here, a tolerating consistency relation with a small t is more tolerant, as correspondences are allowed to be less consistent, than a relation with a t close to 1. A qualitative consistency relation is always considered strict, whereas a quantifiable consistency relation is only strict if all its correspondences have a score of 1.0. This reflects that they dos not tolerate any inconsistencies. When reconsidering the running example and the normalized string similarity (subsubsection 3.3.6), we could classify the quantifiable consistency relation *S* as 0.6-*tolerating*, as each correspondence yields a higher score than 0.6. Simply put, the consistency relation tolerates the inconsistency between the names Component and ComponentImpl. In turn, the quantifiable consistency relation *T* is *strict*, as the score yields 1.0 for the single correspondence. Last but not least, we showcase gradual consistency relations in Figure 9 employing the scoring function *score* introduced in the previous subsubsection. Please note, that we defined gradual consistency regarding the threshold t that all hyper-edges in a quantifiable consistency relation must reach using its scoring function. In this example, *R* could be classified as a *t-tolerating* consistency relation with $t = 0.\overline{6}$. Conversely, *S* could be classified as a *strict* consistency relation. Notably though, any non-quantifiable consistency relation is considered strict, as it has no consistency score and thus cannot be t-tolerating.

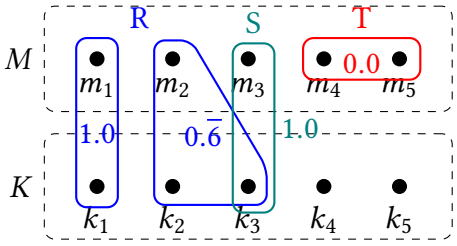


Figure 9: This diagram highlights strict (S) and *t-tolerating* consistency relations (R with $t = 0.\overline{6}$). Each hyper-edge is annotated with its score.

While most engineers consider temporal inconsistencies, i.e., a sequence of inconsistent states between consistent states, as a form of tolerating inconsistencies, we argue that all intermediate states are strictly inconsistent until a consistent state is reached. Instead, we classify consistency relations that are to a certain yet tolerated degree inconsistent as gradually tolerating.

4 Example Scenarios

Henceforth, we present seven MBE scenarios contributed by the domain experts of the focus groups. While we concede that the scenarios might not be representative, we argue that it is almost impossible to collect a set covering all relevant scenarios. Nonetheless, the focus group has collected a diverse set of examples that highlight the different properties of consistency relations. For each scenario, first the purpose of the consistency relation is described, then the classification schema is applied and finally selected consistency relations are classified.

4.1 Multi-View Modeling

In MBE, views on a model allow modelers to see it from different viewpoints [6], such that modelers can focus on a particular aspect. A view might abstract, aggregate, or filter information from its underlying model, thus, reducing the complexity by providing only relevant information for a specific concern. The problem of updating the model of an editable view when the view is changed is denoted the *view-update problem* [8]. Solving this problem for models requires restoring the consistency between the model and its views to ensure the absence of contradicting information [17]. For this purpose, consistency relations are specified between the metamodel and viewtype [21], e.g., by utilizing model transformations. After a view is created, the consistency relation establishes a trace link between elements of the view and elements of the underlying model. As views can be read-only or editable and permit state-based or change-based differencing [59], not all combinations need to employ consistency relations, e.g., read-only or state-based views. Please note that for brevity, we henceforth do not consider views combining multiple models, e.g., [11].

Abstraction A model-view consistency relation can be horizontal or vertical. In the former case, a view only shows selected parts of the underlying model. In the latter case, a view abstracts from the underlying model.

Metalevel Model-view consistency relations are, per definition, intra-metalevel.

Position As trace links connect elements from a view to elements of the underlying model, they classify as coupled-model consistency relations.

Observed Property As the model is projected into a view, depending on the underlying model, the consistency relation observes either structural, behavioral, or both.

Development Phase All consistency relations are intra-phased, as a view belongs to the same phase as its underlying model.

Quantification In general, model-view consistency relations are qualitative. However, [66, 35] propose model differencing to quantify the degree of model-view consistency.

Gradual Consistency Model-view consistency relations are strict, as concurrent changes to the view and model directly lead to an inconsistent view.

The consistency relation between a view and its source model is intra-metalevel, coupled-model, intra-phased, qualitative, and strict. However, the abstraction and observed properties depend on the particular view type.

4.2 Metamodel Evolution and Co-Evolution

Metamodel evolution is the process of applying changes to metamodels. This impacts other artifacts that depend on these metamodels, e.g., instances, transformations, generator templates, and editors. When metamodels evolve, these artifacts have to be co-evolved so that they stay consistent with the metamodels. This has been extensively researched for co-evolution of instances in [26, 10, 23].

Abstraction Artifacts other than metamodels that have to co-evolve are at a lower level of abstraction. Thus, the consistency relation is vertical.

Metalevel For several artefacts, the metalevel classification does not apply (e.g., transformations, generators, and editors). For co-evolution of metamodels and instances, the relation is inter-metalevel.

Position Co-evolution always implies inter-model consistency relations. Typically, these are also coupled-model consistency relations, e.g., [10].

Observed Property Depending on the kind of artifacts, consistency is mainly determined by syntactical properties, such as being a valid instance of a class or a valid word in a language, or semantic properties, for example preserving the desired behavior of model transformations.

Development Phase In most development processes, metamodel design is a separate phase, and metamodel changes are heavy-weight and seldom. Thus, the consistency relation is inter-phase.

Quantification Changes to metamodels can be quantified by the impact on depending artifacts and the effort needed to co-evolve them.

Gradual Consistency The consistency relation is strict, as tolerated metamodel changes would inevitably lead to invalid artifacts, either syntactically or semantically.

The consistency relation between a metamodel and its co-evolving instance [10] is vertical, inter-metalevel, and has coupled models. It observes structural properties, is inter-phase, quantifiable, and strict.

4.3 Sketches and Informal Diagrams

During software development, teams use informal diagrams of software architecture to ease communication with other stakeholders and planning activities [22], e.g., during discussions at a white board. Therefore, diagrams have to be aligned with existing architecture models [19] or with existing source code artifacts [7]. In case of a hand-drawn sketch, the image must first be translated into a machine-readable diagram, e.g., using image recognition approaches. Afterwards, the machine-readable diagram encompassing shapes, lines, and texts is mapped to the existing architecture. Thus, this case covers two types of consistency relations: (1) The *interpretation* between the sketch and the interpreted machine-readable diagram ensures that every sketched element is present in the generated machine-readable diagram. (2) The *mapping* of the machine-readable diagram to the architecture model assigning each shape and line to the corresponding architectural elements. Both consistency relations connect different models to provide tracing information for developers.

Abstraction The interpretation a horizontal consistency relation, since we consider informal but technical sketches. In contrast, mapping is a vertical consistency relation, as the architecture model contains more detailed information, e.g., method names in interfaces omitted in sketches.

Metalevel Consistency of diagrams and architecture models is intra-metalevel, since connects corresponding representations of elements in different artifacts.

Position Typically, both are coupled-model consistency relations.

Observed Property Regarding the observed property, these consistency relations considers structural models, e.g., class diagrams or component diagrams.

Development Phase Since sketches and diagrams can be used in any phase of a project, the consistency relations can be inter- or intra-phase.

Quantification The interpretation is typically quantifiable using confidences, e.g., object detection algorithms using machine learning [60]. Although the properties of the *mapping* are still being researched, we expect that it will be quantifiable with heuristics.

Gradual Consistency Inconsistencies can be tolerated in this case depending on the purpose of the diagram. If the diagram is used to explain an extension of the current system, new elements have no corresponding elements in the current system model. Therefore, they can be tolerated. If the diagram should show a representation of a current system, inconsistencies are not tolerated.

The interpretation is horizontal, intra-metalevel, coupled-model, observes structural properties, can be inter- or intra-phased, is typically quantified and tolerates inconsistencies. In contrast to that, the mapping is vertical.

4.4 Natural Language Descriptions

Natural language (NL) descriptions are present in every software development project [28] and range, among others, from requirements over architecture documentation to source code documentation. For example, software architecture documentation captures and preserves knowledge about the architecture of a system, including knowledge about certain design decisions, e.g., alternatives and reasoning [32, 15]. NL descriptions of a system are a different view on the system and, therefore, need to match to other views like code or architecture models. There are different kinds and reasons for inconsistencies [32, 74], e.g., outdated documentation after system evolution [48] or descriptions differing from other artifacts due to varying interpretation or understanding. Therefore, consistency here means that the interpretation of statements needs to be unambiguous and fit to other artifacts like source code or models. The consistency relations in this case show that, e.g., requirements are implemented correctly or documentation correspond to code. In addition, there are consistency relations within a description artifact [48], e.g., within requirement documents where the consistency relations help avoiding contradictions.

Abstraction There can be both, horizontal abstraction, e.g., between requirements, and vertical abstraction, e.g., between documentation and source code.

Metalevel Most of the time, the consistency relation is intra-level. However, there are cases where, e.g., classes and concrete objects are described, which results in inter-level consistency relations.

Position As there are consistency relations within and between artifacts, there are intra- as well as inter-model consistency relations. Consequently, depending on each concrete case, there can be coupled-model consistency (e.g., between NL documentation of the architecture and architecture model), but this is not the general case.

Observed Property Descriptions can cover structural (e.g., existing components and their orchestration) and behavior aspects (e.g., functionality or performance).

Development Phase NL descriptions like requirements, documented design descriptions, and code documentation are created in every phase. The consistency relation can cover all kinds, intra-phase (e.g., code documentation fits to code) and inter-phase (e.g., requirement is implemented in code correctly).

Quantification In many cases, the consistency relation is qualitative. However, there are quantitative consistency relations. For example, for a prescriptive NL description, there can be a score that states the degree of fulfillment, thus quantifying the consistency relation. For example, there can be a relation between a requirement and one or more classes and the quantification shows how much of the requirement is already realized.

Gradual Consistency Inconsistencies in this case are usually tolerated. For example, inconsistencies based on prescriptive documentation, e.g., requirements, shows the parts that are not yet fully realized. Additionally, inconsistencies are not problematic per se [49] and for each inconsistency, developers can decide to either tolerate or fix it, e.g., based on impact and required effort.

The consistency between NL architecture documentation and source code is vertical, intra-level, coupled-model, observes structural properties, is intra-phase. Sometimes, they are quantifiable and tolerate inconsistencies.

4.5 Composition of Analyses

Sometimes multiple analyses must be employed to show properties of the system [71]. For static security analyses, for instance, static data flow analysis on the software architecture is combined with a data flow analysis on the corresponding implementation to make statements about security aspects of the system. Since not all models cover all aspects of the system, security analyses are forced to make assumptions about missing aspects [62]. When the software architecture does not capture the components' behavior, security leaks by illegal information flows cannot be analyzed [63]. Therefore, for a static architectural security analysis, it is assumed that the behavior does not induce leaks through illegal information flows. Even if the components' behavior was specified, the conformance between the software architecture and its implementation must be assumed [52]. In all cases, architectural models and security assumptions need to be kept consistent with the specifications and implementations to ensure that the system is correctly represented when analyzed. For analysis, the assumptions in a software architecture can be verified by checking the assumed aspect against its implementation [20]. Here, consistency relations between the design artifacts are utilized, e.g., between the software architecture with security information and the implementation with analysis specific annotations. In addition, this consistency relation can be used to transfer modifications in the implementation back to the software architecture if assumptions are affected.

Abstraction The outlined consistency relations are vertical, as they connect elements of an abstract software architecture and concrete source code [20, 52, 62].

Metalevel We classify the consistency relation as intra-metalevel, as all model elements belong to the same metalevel. However, when source code analysis results are included in the consistency relation, it is inter-metalevel.

Position The consistency relations is inter-model yet not a coupled-model consistency, as it connects elements of the software architecture and security specification with the source code if they are fulfilled.

Observed Property As the coupled analysis of architectures and implementations with security annotations involves model elements observing structural, behavioral as well as quality properties, some consistency relations focus on structural properties, on quality properties (i.e., security) or on behavioral properties [51], whereas more general consistency relations observe both structural, behavioral, and quality properties [58].

Development Phase The consistency relations are inter-phased, as they connect architectural elements belonging to specification time with source code at design time.

Quantification The consistency relations in the composition of analysis are qualitative, as any slight inconsistency likely invalidates the analysis results.

Gradual Consistency As no inconsistency are allowed for analysis, the consistency relations are strict.

The consistency relation between a software architecture and the corresponding implementation in the context of security analyses [58] is vertical, intra-metalevel, and inter-model. It observes both structural, behavioral, qualitative properties, and is inter-phase, qualitative, and strict.

4.6 Performance Modeling and Measurement

In agile software development, the performance measurement for future scenarios, e.g., alternative workload, design, deployment, is expensive. The Architecture-based Performance Prediction (AbPP) reduces such costs but requires an accurate architectural performance model. The accuracy of the modeled or extracted performance model can be affected by frequent source code changes during development and adaptations of the system during its operation, e.g., changes in system composition, deployment, or execution environment. The continuous integration of architectural performance models (CIPM) approach [42] addresses this problem by keeping the software system and the corresponding performance model consistent. Since the system monitoring makes detecting system adaptations and system usage possible, CIPM preserves the consistency between the software system, performance model, and measurements to ensures the accuracy of performance predictions [40]. In detail, CIPM combines three consistency relations: (1) between the source code and the performance model to prevent drift or erosion [41]; (2) between the performance model and software architecture extracted from the measurements to reflect the adaptation of the software system at run time [45]; (3) between the source code, the performance model and measurements to ensure the accuracy for predicting the performance of the system [42, 72].

Abstraction All consistency relations are vertical since the performance model is an abstraction of both the source code and measurements.

Metalevel The consistency relations are on multiple metalevels. Although both the performance model and source code are on the model level, measurements are on the instance level. Thus, the first consistency relation is intra-metalevel and the others are inter-metalevel.

Position All consistency relations are coupled-model consistency relations.

Observed Property Since the performance model captures both structural and behavioral properties of the software system, the consistency relations observe both properties. However, while the first two focus on structural properties the last focuses on behavioral properties.

Development Phase While the first consistency relation is intra-phased, the others are inter-phased between development, testing, and operations.

Quantification All consistency relations are quantifiable using different scores [42, 45], e.g., the Wasserstein Distance [43] quantifies the difference between the measured performance and its prediction.

Gradual Consistency While the first two consistency relations are strict, the third is always tolerating, as both performance predictions and measurements have margins of error.

The consistency relations between software architecture, performance model, and corresponding measurements are vertical, intra-(1) or inter-metalevel (2,3), and coupled-model. They focus on structural (1,2) or behavioral (3) properties. They are intra-(1) or inter-phase (2,3), quantifiable, and either strict (1,2) or tolerating (3).

4.7 Variability Management

Variability denotes *"the ability to derive different products from a common set of artifacts"* [4]. Software product line (SPL) engineering [29, 54, 12] is an established approach to systematically engineer and manage the reusability and extensibility of software by utilizing an explicit variability model to configure and combine a set of reusable artifacts to derive an individual product variant [54]. During SPL engineering, the variability demanded in the variability model must correspond to the variability enabled by reusable artifacts, whereas inconsistencies between demanded and enabled variability can lead to broken products or unrealized potential. During SPL evolution, changes to the variability model might invalidate previously valid configurations and changes to reusable artifacts can impact the enabled variability. While [1] identified 14 types of inconsistencies in SPLs, in general, these consistency relations ensure that the managed variability is equally reflected in the variability model and valid configurations in the *problem space*, e.g., [46, 25, 37], as well as the reusable artifacts and derived product variants in the *solution space*, e.g., [2, 31, 9].

Abstraction While consistency relations within a space are usually horizontal [9, 47, 25], consistency relations between problem and solution space are vertical [2, 16].

Metalevel Usually, the consistency relations are intra-metalevel, as variability and its realization co-exists on the same metalevel, with the exception of delta-oriented approaches, that elevate variability to the metamodel level [53].

Position Although most consistency relations are coupled-model consistencies such as variability model–configuration, configuration–derived product variant, specific intra-model consistencies exist, e.g., consistency of variability models [25] or well-formedness of a product variants [31].

Observed Property As the problem and solution space describe the variability of both structural and behavioral elements, most consistency relations usually observe both structural and behavioral properties of the variable system.

Development Phase Considering the feature-oriented SPL engineering process [4], almost all consistency relations are inter-phased, with the exception of the intra-model consistency of variability models.

Quantification Although most consistency relations in SPL engineering are not quantified, several approaches determine the impact of changes or minimal number of repair operations to guide users towards a consistent SPL, e.g., [47].

Gradual Consistency In SPL, all consistency relations are strict, as inconsistencies lead to unrealized variability and/or broken product variants.

Although exceptions exist in multiple classes, most consistency relations in SPLs are either horizontal or vertical, intra-metalevel, coupled-model, observe both structural and behavioral properties, are inter-phased, qualitative and strict.

4.8 Summary

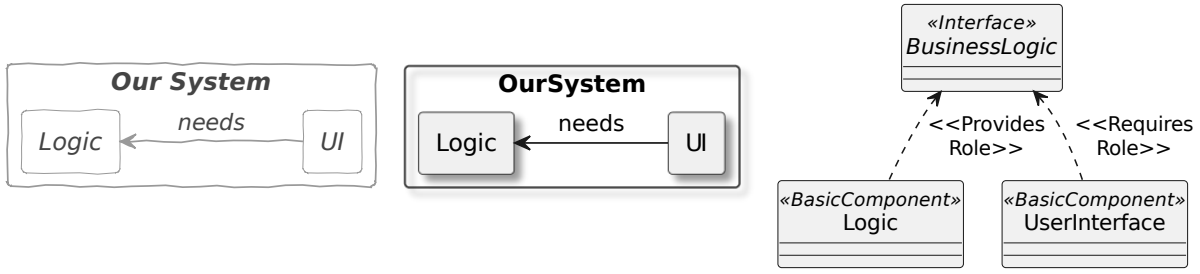
After applying the classification schema to the above MBE scenarios, Table 1 summarizes the classification of the main consistency relation for each scenario. Here, multiple classifications for a relation are separated by a slash (/). In case of Sketches and Informal Diagrams and Performance Modeling, the above classification was applied to the specific consistency relations that have been identified. However, as there are a multitude of consistency relations in Variability Management and NL Descriptions alone, we refrained from a classification of each individual consistency relation.

While most classified consistency relations are vertical, we have found horizontal consistency relations in Sketches and Informal Diagrams, NL Descriptions, and Variability. Likewise, most consistency relations stay within a metalevel, however, consistency relations in Metamodel Evolution, in Performance Modeling and in NL Descriptions are inter-metalevel. Interestingly, we have found that most of our scenarios contain coupled-model consistency relations, albeit we have identified particular intra-model and inter-model consistency relations in Variability management and Composition of Analyses, respectively. Similar surprising was the wide range of properties observed by consistency relations, such that some focus on structural or behavioral properties while some combine both. In particular, for the Composition of Analyses, we have identified *quality* as another observable property. Moreover, we have counted equally many inter-phased and intra-phased consistency relations. In regard to Quantification and Gradual Consistency, we have discovered that most consistency relations are quantifiable with specific scoring functions, whereas only Multi-View Modeling and Composition of Analyses feature qualitative consistency relations. Nonetheless, while most consistency relations are strict, Sketches and Informal Diagrams and Performance Modeling feature tolerating consistency relations.

	Abstraction	Metalevel	Position	Observed Property	Development Phases	Quantification	Gradual Consistency
Multi-View Modeling	h/v	a	c	s/b	a	l	i
Metamodel Evolution	v	e	e/c	s/b	e	q	i
Sketches and Informal Diagrams	h v	a	c	s	a/e	q	t/i
NL Descriptions	h/v	a/e	a/e	s/b	a/e	q	t
Composition of Analyses	v	a	e	s/b/q	e	l	i
Performance Modeling	v	a e e	c	s s b	a e e	q	i i t
Variability	h/v	a	c/a	s/b	a/e	l/q	i

Table 1: Mapping of Classifiers to MBE Scenarios

h/v:	horizontal/vertical	a/e:	intra-/inter-(level/model/phase)
c:	coupled-model	s/b/q:	structural/behavioral/quality
l/q:	qualitative/quantitative	t/i:	tolerating/strict



(a) Hand-drawn sketch (b) Machine readable diagram (c) Visualized Architecture Model

Figure 10: A hand drawn sketch (a) a machine readable version (b) and a corresponding Architecture Model (Palladio Component Model) (c) for a small example architecture.

In conclusion, Table 1 shows that each consistency relation could be classified and each property of our classification appears at least once. While some properties occur more frequent than others, for most properties, we found at least three out of seven occurrences. Exceptions are intra-model relations in Position that only occurs twice and quality in Observed Property that only appears once.

5 Illustrative Application to an Example Scenario

To highlight how an example scenario was investigated, in this section we illustrate the application of the formal definitions to classify the consistency relations found in the scenario. For this illustration, we again focus on the example scenario of Sketches and Informal Diagrams (discussed in subsection 4.3).

5.1 Example

Figure 10 shows an example based on Sketches and Informal Diagrams. It encompasses a hand-drawn sketch of a software architecture, its machine readable version that is created by some object detection (e.g., machine learned model), and a formal architecture model (e.g., Palladio Component Model) that is created by a software architect.

The hand-drawn sketch is depicted in Figure 10a. It consists of the following elements: First, there is the caption of the sketch, which is Our System. Second, there are two sketched boxes with the labels Logic and UI. Finally, there is the drawn arrow between the two boxes, which is labeled with needs.

To create a machine readable version of the sketch, we assume a machine learning model that detects boxes and arrows in images. The result of this interpretation is depicted in Figure 10b. As you can see, the machine readable version is not perfectly detected as the caption is missing a space. The other elements have been detected correctly. Since, we used a machine learning model, we can also provide a confidence score for each detected element. In this example,

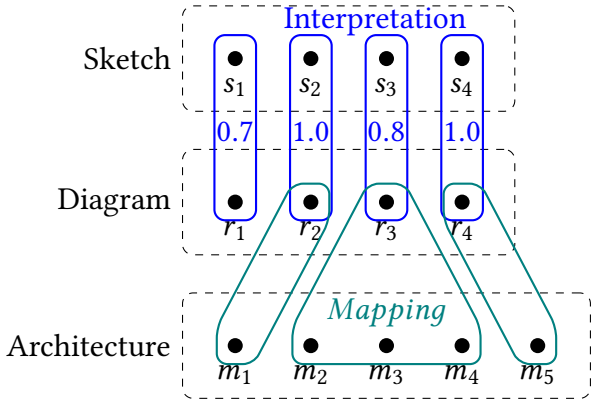


Figure 11: Illustration of the Interpretation (blue) and Mapping (teal) consistency relations between model elements of the sketch and the diagram as well as the diagram and the architecture.

we assume that the confidence for the detected boxes are 1.0, the confidence for the detected dependency-arrow is 0.8, and the confidence for the detected caption is 0.7. For simplicity, we will use the confidence scores directly to quantify the consistency relation between the sketch and the machine readable diagram.

The final model we need for this example is the formal architecture model. In this example, we use the Palladio Component Model (PCM) depicted in Figure 10c. The architecture consists of two BasicComponents, namely Logic and UserInterface. The UserInterface has a dependency to some implementation of the BusinessLogic interface, which is, in turn, provided by the Logic. In total, the architecture model has five elements, i.e., two BasicComponents, two Roles, and one Interface.

While the consistency relation between the sketch and the machine readable diagram in this example is a 1:1 mapping, the consistency relation between the machine readable diagram and the formal architecture model is more complex. The two detected boxes in the machine readable diagram are mapped to the two BasicComponents in the architecture. The detected arrow is mapped to the dependency between the two BasicComponents. This dependency corresponds to the two Roles and the Interface. The caption of the machine readable diagram is not mapped to any element in the PCM. We visualize the consistency relations as hyper-graph in Figure 11.

The elements of the sketch are the caption s_1, the box Logic s_2, the arrow s_3, and the box UI s_4. The elements of the machine readable diagram are the caption r_1, the box Logic r_2, the dependency-arrow r_3, and the box UI r_4. Finally, the elements of the architecture model are the BasicComponent Logic $m1$, the Interface BusinessLogic m_2, the two Roles m_3, m_4, and the BasicComponent UserInterface m_5.

The *interpretation* consistency relation connects the sketch with the machine readable diagram and is drawn in blue. Since, we use the confidences for quantification, the values are directly marked on the edges. The *mapping* consistency relation connects the machine readable diagram with the architecture model and is drawn in teal.

5.2 Interpretation

Since the sketch and machine readable diagram do not contain elements that abstract from elements in one another, the consistency relation is *horizontal*. Furthermore, the elements of both models are on the same metalevel. Therefore, the consistency relation is intra-level. Every edge of the interpretation connects elements of both models, therefore the consistency relation is *coupled* w.r.t. *Position*. Additionally, the consistency relation *focuses on structural properties* since the elements of both models are structural elements. The development phase is not set for this example, but we assume that the sketches and also the diagrams belong to an early design phase in this scenario; if this would be the case, the consistency relation would be *intra-phase*. Regarding quantification, we can see that the confidence values can be used as scores for the hyper-edges of the interpretation. Therefore, the consistency relation is *quantifiable*. Finally, the interpretation is *t-tolerating* with $t = 0.6$ since the confidence values of the correct edges are greater than t.

5.3 Mapping

Since the dependency-arrow in the machine readable diagram is an abstraction of the interface and roles in the architecture model, the consistency relation *mapping* is *vertical*. Nevertheless, the models are on the same metalevel and therefore, the consistency relation is intra-level. The consistency relation is *coupled* w.r.t. *Position*, since each hyper-edge contains at least one element of the machine readable diagram and one element of the architecture model. The consistency relation *focuses on structural properties* since the elements of both models are structural elements. Again, it is not clear in which development phase the models are created, but if we imagine that the diagram is created in an early design phase and the architecture model in a later design phase, the consistency relation would be *inter-phase*. Regarding quantification, heuristics could provide some confidence values for the mapping here as well. If such confidences exist, they could be used as scores for the hyper-edges of the mapping analogous to the interpretation.

In summary, we have presented a detailed application of our classification to two consistency relations from the sketches and informal diagram domain. We have shown how the classification is applied and able to capture the characteristic differences of the consistency relations.

6 Discussion and Findings

We have established a focus group to elucidate the notion of consistency relations and their properties. These properties were applied by the domain experts to classify seven of their MBE scenarios. Consequently, we can answer our research questions:

(RQ1) *How can consistency relations be precisely defined?* While we have found existing, precise descriptions of consistency and its properties for specific use cases and scenarios, none of them provided a generally applicable formalization and classification of consistency relations (cf. subsection 2.2). Even though [57] provide some formalization, they focus on

database applications. [73] do not provide a formalization for their consistency classes. In contrast to them, our classification provides more fine grained properties to describe the scope of consistency relations. Moreover, they do not fully separate between consistency relations and consistency preservation or repair. Unlike their work, we argue that for precisely describing consistency relations it is necessary to define them among model elements. This novel perspective was crucial to be able to formally define both consistency relations and classifying properties when discussing them in the focus groups. Thus, in this paper, we have provided a comprehensible formalization that enables further research of consistency relations in MBE. As the domain experts could easily apply the classification schema to their MBE scenario, we gained evidence that the classification schema is precise, comprehensive, and generally applicable.

(RQ2) *What use case independent properties classify consistency relations?* Based on existing classification schemes, we have identified seven generally applicable properties of consistency relations: Abstraction, Metalevel, Position, Observed Property, Development Phases, Quantification and Gradual Consistency. From the formalization of the classification schema, it follows that all classifiers are independent except t-tolerating, which requires a quantifiable consistency relation. Besides that, we have observed dependencies between abstraction and Metalevel as well as Development Phases. In all observed cases, an inter-metalevel or inter-phase consistency relation is a vertical abstraction. Regardless, six of seven properties are *orthogonal*. As the classification schema could be easily applied to seven different MBE scenarios, we maintain that our classification schema is generally applicable to MBE scenarios. Notably, we have identified two additional properties. *Dominance* describes consistency relations with certain models that represent the ground truth for other models. For example, measurements of the system provide the ground truth to which a performance prediction must adhere to. *Description* distinguishes between prescriptive and descriptive consistency relations [33]. Both properties are out of scope, as they consider consistency preservation and repair, but not consistency relations themselves. Besides that, while consistency relations between an analytical model and measurements exist in Performance Modeling, it is still unclear whether this holds true for all analytical or data-defined models. In general, Table 1 shows, that each consistency relation of our scenarios has been classified and each property of our classification is needed. In future work we will investigate a larger corpus of consistency relations to gather evidence on the sufficiency, distribution, and correlation of the classifying properties.

7 Threats to Validity

Here, we discuss how we addressed threats to validity. To ensure the **construct validity** of our classification schema, we opted to formalize both consistency relations and its classifying properties in addition to regular descriptions. This avoided ambiguities and ensured orthogonality of the classification schema. The only exception is t-tolerance which depends on a quantifiable consistency relation. We retrieved the relevant classifying properties from existing related classification schemes and our focus group. The applicability was indicated within our

focus group, whereas each expert performed the classification of her/his scenario individually. While we focused on the classification's structural suitability, we have not yet evaluated our classification schema with taxonomy evaluation methods, e.g., [30], this will be future work.

Regarding the **internal validity**, we first identified contemporary definitions of consistency and classification schemes for consistency focusing on peer-reviewed publications within and beyond the field of MBE. We established focus groups with experts in MBE from academia to elucidate a fine-grained notion of consistency relations and classifying properties. For the scenario selection, we encouraged the domain experts to propose their own MBE scenario, while we ensured that each scenario was distinct. For the classification phase, we provided a brief introduction to the formalized classification schema to the focus group, such that each domain expert could individually classify the consistency relations in her/his scenario. The resulting classification was then reviewed by the group moderators, to uncover misunderstandings. Beyond that, we cannot rule out annotator bias. Please note that as the description and classification of each scenario was done by a domain expert, we added each of them as co-author.

To improve **external validity**, by establishing a focus group, we aimed to uncover a fine-grained definition of consistency. However, as the MBE scenarios were proposed by the domain experts, we cannot consider them as a representative subset, especially, as neither analytical models nor data-derived models are considered. While this significantly limits the generalizability of our findings regarding the classified consistency relations, we maintain that our definition of consistency relations and our classification schema is generally applicable to the MBE domain. Nevertheless, a large scale study and analysis of MBE scenarios should be conducted in future works to show the generalizability of our results.

8 Conclusion

In this paper, we define a precise and fine-grained notion of consistency relations for the MBE community, as there were neither an established common understanding nor use-case-independent definitions. To remedy this, we established focus groups and elucidated fine-grained, notion of consistency relations over model elements. From this notion, we derived our formalized classification schema of consistency relations encompassing seven classifying properties based on set theory and hyper-graphs. We have illustrated the applicability of our classification schema by having experts apply it to their MBE scenarios. Although these scenarios are widely different, we could completely and distinctly classify each identified consistency relation. In general, each property class is needed, as its class occurred at least once in these scenarios.

In conclusion, we argue that our classification schema provides a precise fine-grained definition and vocabulary for the MBE community to distinguish consistency relations. This, in turn, can help to raise awareness for the different consistency notions and facilitate a common ground for understanding, discussing, and finding commonalities of consistency in the MBE community. In future work, we will investigate a wider range of consistency relations to classify and evaluate our classification schema according to [30].

References

[1] Sofia Ananieva. "Consistent Management of Variability in Space and Time". In: *Proceedings of the 25th ACM International Systems and Software Product Line Conference – Volume B*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 7–12. DOI: 10.1145/3461002.3473067.

[2] Sofia Ananieva, Thomas Kühn, and Ralf Reussner. "Preserving Consistency of Interrelated Models during View-Based Evolution of Variable Systems". In: *Proceedings of the 21st ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. GPCE 2022. Auckland, New Zealand: Association for Computing Machinery, 2022, pp. 148–163. DOI: 10.1145/3564719.3568685.

[3] Michał Antkiewicz and Krzysztof Czarnecki. "Design Space of Heterogeneous Synchronization". In: *Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007. Revised Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 3–46. DOI: 10.1007/978-3-540-88643-3_1.

[4] Sven Apel et al. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013.

[5] Colin Atkinson and Thomas Kühne. Rearchitecting the UML Infrastructure. In: *ACM Trans. Model. Comput. Simul.* 12.4 (Oct. 2002), pp. 290–321. DOI: 10.1145/643120.643123.

[6] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. "Orthographic Software Modeling: A Practical Approach to View-Based Development". In: *Evaluation of Novel Approaches to Software Engineering*. Vol. 69. Communications in Computer and Information Science. Berlin, Heidelberg: Springer, 2010, pp. 206–219. DOI: 10.1007/978-3-642-14819-4_15.

[7] Sebastian Baltes, Peter Schmitz, and Stephan Diehl. "Linking sketches and diagrams to source code artifacts". en. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Hong Kong China: ACM, Nov. 2014, pp. 743–746. DOI: 10.1145/2635868.2661672.

[8] F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. In: *ACM Trans. Database Syst.* 6.4 (Dec. 1981), pp. 557–575. DOI: 10.1145/319628.319634.

[9] Paul Maximilian Bittner et al. "Feature Trace Recording". In: *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2021. Athens, Greece: Association for Computing Machinery, 2021, pp. 1007–1020. DOI: 10.1145/3468264.3468531.

[10] Erik Burger and Boris Gruschko. "A Change Metamodel for the Evolution of MOF-Based Metamodels". In: *Proceedings of Modellierung 2010*. Vol. P-161. GI-LNI. Klagenfurt, Austria, Mar. 2010, pp. 285–300.

[11] Erik Burger et al. View-based model-driven software development with ModelJoin. In: *Software & Systems Modeling* 15 (2016), pp. 473–496. DOI: 10.1007/s10270-014-0413-5.

[12] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.

[13] Gregor Engels et al. "A Methodology for Specifying and Analyzing Consistency of Object-Oriented Behavioral Models". In: *Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ESEC/FSE-9. Vienna, Austria: Association for Computing Machinery, 2001, pp. 186–195. DOI: 10.1145/503209.503235.

[14] K. P. Eswaran et al. The Notions of Consistency and Predicate Locks in a Database System. In: *Commun. ACM* 19.11 (Nov. 1976), pp. 624–633. DOI: 10.1145/360363.360369.

[15] R. Farenhorst and H. van Vliet. "Understanding how to support architects in sharing knowledge". In: *ICSE Workshop on Sharing and Reusing Architectural Knowledge*. 2009, pp. 17–24. DOI: 10.1109/SHARK.2009.5069111.

[16] Kevin Feichtinger et al. Guiding feature model evolution by lifting code-level dependencies. In: *Journal of Computer Languages* 63 (2021), p. 101034. DOI: https://doi.org/10.1016/j.cola.2021.101034.

[17] J. Nathan Foster et al. Combinators for Bi-Directional Tree Transformations: A Linguistic Approach to the View Update Problem. In: *SIGPLAN Not.* 40.1 (Jan. 2005), pp. 233–246. DOI: 10.1145/1047659.1040325.

[18] Michael J. Franklin, Michael J. Carey, and Miron Livny. Transactional Client-Server Cache Consistency: Alternatives and Performance. In: *ACM Trans. Database Syst.* 22.3 (Sept. 1997), pp. 315–363. DOI: 10.1145/261124.261125.

[19] Dominik Fuchß. "Sketches and Natural Language in Agile Modeling". In: *Companion Proceedings of the 15th European Conference on Software Architecture (ECSA-C)*. Vol. 2978. CEUR Workshop Proceedings. 2021. DOI: 10.5445/IR/1000139435.

[20] Johannes Geismann, Bastian Haverkamp, and Eric Bodden. "Ensuring threat-model assumptions by using static code analyses". In: *ECSA 2021 Companion Volume, Virtual (originally: Växjö, Sweden), 13-17 September, 2021*. Vol. 2978. CEUR Workshop Proceedings. CEUR-WS.org, 2021.

[21] Thomas Goldschmidt, Steffen Becker, and Erik Burger. "Towards a Tool-Oriented Taxonomy of View-Based Modelling". In: *Proceedings of the Modellierung 2012*. Vol. P-201. GI-Edition – Lecture Notes in Informatics (LNI). Bamberg: Gesellschaft für Informatik e.V. (GI), Mar. 2012, pp. 59–74.

[22] Wilhelm Hasselbring. "Software Architecture: Past, Present, Future". en. In: *The Essence of Software Engineering*. Cham: Springer International Publishing, 2018, pp. 169–184. DOI: 10.1007/978-3-319-73897-0_10.

[23] R. Hebig, D. E. Khelladi, and R. Bendraou. Approaches to Co-Evolution of Metamodels and Models: A Survey. In: *IEEE Transactions on Software Engineering* 43.5 (May 2017), pp. 396–414. DOI: 10.1109/TSE.2016.2610424.

[24] Mats Per Erik Heimdahl and Nancy G. Leveson. Completeness and consistency in hierarchical state-based requirements. In: *IEEE transactions on Software Engineering* 22.6 (1996), pp. 363–377. DOI: 10.1109/32.508311.

[25] Marc Hentze et al. “Hyper Explanations for Feature-Model Defect Analysis”. In: *VaMoS’21: 15th International Working Conference on Variability Modelling of Software-Intensive Systems, Virtual Event / Krems, Austria, February 9-11, 2021*. ACM, 2021, 14:1–14:9. DOI: 10.1145/3442391.3442406.

[26] Markus Herrmannsdörfer, Sander D. Vermolen, and Guido Wachsmuth. “An extensive catalog of operators for the coupled evolution of metamodels and models”. In: *Proceedings of the Third international conference on Software language engineering*. SLE’10. Berlin/Heidelberg: Springer, 2011, pp. 163–182. DOI: 10.5555/1964571.1964585.

[27] Zbigniew Huzar et al. “Consistency Problems in UML-Based Software Development”. In: *UML Modeling Languages and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 1–12.

[28] Michael Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press/Addison-Wesley Publishing Co., 1995.

[29] Kyo C. Kang et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Tech. rep. Carnegie-Mellon University Software Engineering Institute, Nov. 1990. DOI: 10.1.1.606.7899.

[30] Angelika Kaplan et al. “Introducing an Evaluation Method for Taxonomies”. In: *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*. EASE ’22. Gothenburg, Sweden: Association for Computing Machinery, 2022, pp. 311–316. DOI: 10.1145/3530019.3535305.

[31] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. “Consistency-preserving edit scripts in model versioning”. In: *Proceedings of the 28th International Conference on Automated Software Engineering (ASE’13), Silicon Valley, USA*. IEEE, 2013, pp. 191–201. DOI: 10.1109/ASE.2013.6693079.

[32] Jan Keim and Anne Koziolek. “Towards Consistency Checking Between Software Architecture and Informal Documentation”. In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. Mar. 2019, pp. 250–253. DOI: 10.1109/ICSA-C.2019.00052.

[33] Heiko Klare. “Building Transformation Networks for Consistent Evolution of Interrelated Models”. PhD thesis. Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT), 2021. 428 pp. DOI: 10.5445/IR/1000133724.

[34] Heiko Klare and Joshua Gleitze. “Commonalities for preserving consistency of multiple models”. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2019, pp. 371–378. DOI: 10.1109/MODELS-C.2019.00058.

[35] Dimitrios Kolovos et al. “Different models for model matching: An analysis of approaches to support model differencing”. In: *2009 ICSE Workshop on Comparison and Versioning of Software Models*. 2009 ICSE Workshop on Comparison and Versioning of Software Models. May 2009, pp. 1–6. DOI: 10.1109/CVSM.2009.5071714.

[36] Jyrki Kontio, Johanna Bragge, and Laura Lehtola. The Focus Group Method as an Empirical Tool in Software Engineering. In: *Guide to Advanced Empirical Software Engineering*. Ed. by Forrest Shull, Janice Singer, and Dag I. K. Sjøberg. London: Springer London, 2008, pp. 93–116. DOI: 10.1007/978-1-84800-044-5_4.

[37] Matthias Kowal, Sofia Ananieva, and Thomas Thüm. "Explaining Anomalies in Feature Models". In: *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*. GPCE 2016. Amsterdam, Netherlands: Association for Computing Machinery, 2016, pp. 132–143. DOI: 10.1145/2993236.2993248.

[38] Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. In: *IEEE Transactions on Computers* C-28.9 (1979), pp. 690–691. DOI: 10.1109/TC.1979.1675439.

[39] Ueli Maurer. "Towards a Theory of Consistency Primitives". In: *Distributed Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 379–389.

[40] Manar Mazkatli and Anne Koziolek. "Continuous Integration of Performance Model". In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. ICPE '18. Berlin, Germany: ACM, 2018, pp. 153–158. DOI: 10.1145/3185768.3186285.

[41] Manar Mazkatli et al. Continuous Integration of Architectural Performance Models with Parametric Dependencies – The CIPM Approach. In: *submitted to Automated Software Engineering Journal* (2023). DOI: 10.5445/IR/1000151086.

[42] Manar Mazkatli et al. "Incremental Calibration of Architectural Performance Models with Parametric Dependencies". In: *IEEE International Conference on Software Architecture (ICSA 2020)*. Salvador, Brazil, 2020, pp. 23–34. DOI: 10.1109/ICSA47634.2020.00011.

[43] Facundo Mémoli. Gromov-Wasserstein Distances and the Metric Approach to Object Matching. In: *Foundations of Computational Mathematics* 11.4 (2011), pp. 417–487. DOI: 10.1007/s10208-011-9093-5.

[44] Parastoo Mohagheghi, Vegard Dehlen, and Tor Neple. Definitions and approaches to model quality in model-based software development–A review of literature. In: *Information and Software Technology* 51.12 (2009), pp. 1646–1669. DOI: 10.1016/j.infsof.2009.04.004.

[45] David Monschein et al. "Enabling Consistency between Software Artefacts for Software Adaption and Evolution". In: *2021 IEEE 18th International Conference on Software Architecture (ICSA)*. 2021, pp. 1–12. DOI: 10.1109/ICSA51549.2021.00009.

[46] Michael Nieke, Christoph Seidl, and Thomas Thüm. "Back to the Future: Avoiding Paradoxes in Feature-Model Evolution". In: *Proceedings of the 22nd International Systems and Software Product Line Conference - Volume 2*. SPLC '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, pp. 48–51. DOI: 10.1145/3236405.3237201.

[47] Michael Nieke et al. Guiding the evolution of product-line configurations. In: *Software and Systems Modeling* (2021). DOI: doi.org/10.1007/s10270-021-00906-w.

[48] B. Nuseibeh, S. Easterbrook, and A. Russo. Leveraging inconsistency in software development. In: *Computer* 33.4 (2000), pp. 24–29. DOI: 10.1109/2.839317.

[49] Bashar Nuseibeh, Steve Easterbrook, and Alessandra Russo. Making inconsistency respectable in software development. In: *Journal of Systems and Software* 58.2 (2001), pp. 171–180. DOI: https://doi.org/10.1016/S0164-1212(01)00036-X.

[50] OMG. *Model Driven Architecture (MDA): Guide Revision 2.0 of MDA Guide Version 1.0. 1 (12 June 2003)*. 2014.

[51] Sven Peldszus et al. "Secure data-flow compliance checks between models and code based on automated mappings". In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2019, pp. 23–33. DOI: 10.1109/MODELS.2019.00-18.

[52] Sven Matthias Peldszus. Static Security Compliance Checks. In: *Security Compliance in Model-driven Development of Software Systems in Presence of Long-Term Evolution and Variants*. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, pp. 165–219. DOI: 10.1007/978-3-658-37665-9_8.

[53] Christopher Pietsch et al. "Delta-oriented development of model-based software product lines with DeltaEcore and SiPL: A comparison". In: *Model Management and Analytics for Large Scale Systems*. Elsevier, 2020, pp. 167–201. DOI: 10.1016/B978-0-12-816649-9.00017-X.

[54] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[55] Ahana Pradhan and Rushikesh Krishnakant Joshi. A Taxonomy of Consistency Models in Dynamic Migration of Business Processes. In: *IEEE Transactions on Services Computing* 11.3 (2018), pp. 562–579. DOI: 10.1109/TSC.2017.2735413.

[56] Alexander Prestel and Charles N. Delzell. *Mathematical Logic and Model Theory*. Springer London, 2011. DOI: 10.1007/978-1-4471-2176-3.

[57] Krithi Ramamritham and Panos K. Chrysanthis. A taxonomy of correctness criteria in database applications. In: *The VLDB Journal* 5.1 (Jan. 1996), pp. 85–97. DOI: 10.1007/s007780050017.

[58] Frederik Reiche et al. *Model-driven Quantification of Correctness with Palladio and KeY*. Tech. rep. Karlsruher Institut für Technologie (KIT), 2021. DOI: 10.5445/IR/1000128855.

[59] Timur Sağlam and Thomas Kühn. "Towards the Co-Evolution of Models and Artefacts of Industrial Tools Through External Views". In: *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Fukuoka, J, October 10-15, 2021*. 24th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems. MODELS 2021 (Online, Oct. 10–15, 2021). IEEEXplore, 2021, pp. 410–416. DOI: 10.1109/MODELS-C53483.2021.00064.

[60] Bernhard Schäfer, Margret Keuper, and Heiner Stuckenschmidt. Arrow R-CNN for handwritten diagram recognition. en. In: *International Journal on Document Analysis and Recognition (IJDAR)* (Feb. 2021). DOI: 10.1007/s10032-020-00361-1.

[61] Andreas Schönberger and Guido Wirtz. "Taxonomy on Consistency Requirements in the Business Process Integration Context." In: *SEKE*. 2008, pp. 593–598. DOI: 10.1.1.527.2969.

[62] Sophie Schulz et al. "Continuous Secure Software Development and Analysis". In: *Proceedings of Symposium on Software Performance 2021*. SSP'21. Nov. 2021. DOI: 10.5445/IR/1000143320.

[63] Stephan Seifermann et al. Detecting Violations of Access Control and Information Flow Policies in Data Flow Diagrams. In: *The Journal of Systems and Software* 184 (2022). DOI: 10.1016/j.jss.2021.111138.

[64] Marc Shapiro and Pierre Sutra. Database Consistency Models. In: *Encyclopedia of Big Data Technologies*. Cham: Springer International Publishing, 2018, pp. 1–11. DOI: 10.1007/978-3-319-63962-8_203-1.

[65] Herbert Stachowiak. *Allgemeine Modelltheorie*. Wien: Springer Verlag, 1973.

[66] Matthew Stephan and James R. Cordy. "A Survey of Model Comparison Approaches and Applications". In: *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development - Volume 1: MODELSWARD,* INSTICC. SciTePress, 2013, pp. 265–277. DOI: 10.5220/0004311102650277.

[67] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. In: *Software and Systems Modeling* 9.1 (Jan. 2010), pp. 7–20. DOI: 10.1007/s10270-008-0109-9.

[68] Perdita Stevens. "Bidirectionally Tolerating Inconsistency: Partial Transformations". In: *17th International Conference on Fundamental Approaches to Software Engineering (FASE 2014)*. Vol. 8411. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 32–46. DOI: 10.1007/978-3-642-54804-8_3.

[69] Perdita Stevens. Maintaining consistency in networks of models: bidirectional transformations in the large. In: *Software and Systems Modeling* 19.1 (Jan. 2020), pp. 39–65. DOI: 10.1007/s10270-019-00736-x.

[70] Patrick Stünkel et al. Multi-Model Evolution through Model Repair. In: *Journal of Object Technology* 20.1 (Jan. 2021). Workshop on Models and Evolution (ME 2020), 1:1–25. DOI: 10.5381/jot.2021.20.1.a2.

[71] Carolyn Talcott et al. Composition of Languages, Models, and Analyses. In: *Composing Model-Based Analysis Tools*. Cham: Springer International Publishing, 2021, pp. 45–70. DOI: 10.1007/978-3-030-81915-6_4.

[72] Sonya Voneva et al. "Optimizing Parametric Dependencies for Incremental Performance Model Extraction". In: *Software Architecture*. Cham: Springer International Publishing, 2020, pp. 228–240.

[73] Nils Weidmann, Suganya Kannan, and Anthony Anjorin. *Tolerance in Model-Driven Engineering: A Systematic Literature Review with Model-Driven Tool Support*. Unpublished. 2021.

[74] Rebekka Wohlrab et al. "Improving the Consistency and Usefulness of Architecture Descriptions: Guidelines for Architects". In: *2019 IEEE International Conference on Software Architecture (ICSA)*. Mar. 2019, pp. 151–160. DOI: 10.1109/ICSA.2019.00024.

[75] Dengyong Zhou et al. "Learning with local and global consistency". In: *Advances in neural information processing systems*. 2004, pp. 321–328.