JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Evaluating Model Differencing for the Consistency Preservation of State-based Views

Jan Willem Wittler[1], **Timur Sağlam**[1], **and Thomas Kühn**[2]
[1]Karlsruhe Institute of Technology, Germany
[2]Martin Luther University Halle-Wittenberg, Germany

ABSTRACT While developers and users of modern software systems usually only need to interact with a specific part of the system at a time, they are hindered by the ever-increasing complexity of the entire system. Views are projections of underlying models and can be employed to abstract from that complexity. When a view is modified, the changes must be propagated back into the underlying model without overriding simultaneous modifications. Hence, the view needs to provide a fine-grained sequence of changes to update the model minimally invasively. Such fine-grained changes are often unavailable for views that integrate with existing workflows and tools. To this end, model differencing approaches can be leveraged to compare two states of a view and derive an estimated change sequence. However, these model differencing approaches are not intended to operate with views, as their correctness is judged solely by comparing the input models. For views, the changes are derived from the view states, but the correctness depends on the underlying model. This work introduces a refined notion of correctness for change sequences in the context of model-view consistency. Furthermore, we evaluate state-of-the-art model differencing regarding model-view consistency. Our results show that model differencing largely performs very well. However, incorrect change sequences were derived for two common refactoring operation types, leading to an incorrect model state. These types can be easily reproduced and are likely to occur in practice. By considering our change sequence properties in the view type design, incorrect change sequences can be detected and semi-automatically repaired to prevent such incorrect model states.

KEYWORDS Model-view consistency, Model differencing, Model comparison, View-update problem, Change sequence, View-based development.

1. Introduction

Modern software systems have increased in their complexity and model vast domain knowledge (Murer et al. 2011). With the paradigm of model-driven engineering, this knowledge is modeled explicitly (Kent 2002). Interacting with such complex and large-scale models, especially in a collaborative and agile way, is challenging. Whether users or developers, individual stakeholders usually only want access to a subset of the model to address a specific concern. Moreover, they typically benefit from working on a subset, which hides some underlying complexity and allows them to focus strictly on their task. Moreover, some stakeholders might only have limited access to the model or might be prohibited from changing parts of it. *Projective views* allow for defining perspectives on the underlying model, such that stakeholders can see the model from different *viewpoints* (Atkinson et al. 2010). A view is projected from a model by instantiating a *view type* on the model (Goldschmidt et al. 2012). Thus, a view type represents the metamodel of the view, describing the elements and relations the view can contain. Following these definitions, views are projected abstractions from the entirety of the modeled information and its complexity, making them an ideal solution to manage how stakeholders interact with the model (Brunelière et al. 2018; Cicchetti et al. 2019). Additionally, defining view types ensures that the stakeholders' access to the model can be restricted. However, the system evolves during its use, and model-view consistency must be retained. When the view is modified, the changes eventually must be propagated into the

JOT reference format:
Jan Willem Wittler, Timur Sağlam, and Thomas Kühn. *Evaluating Model Differencing for the Consistency Preservation of State-based Views*. Journal of Object Technology. Vol. 22, No. 2, 2023. http://dx.doi.org/10.5381/jot.2023.22.2.a4

underlying model to reflect the modifications. Vice versa, when the underlying model is modified, existing views need to be updated to be consistent with the underlying model again. The problem of updating the underlying model upon view modification while re-generating the same view from the updated model is called the *view-update problem* (Bancilhon & Spyratos 1981; Foster et al. 2005). However, as the model might already contain new changes, propagating the entire modified view state may override the model with outdated information from the view. A better approach is to update the model inductively and thus precipitated by fine-grained modifications to the view. This is called *delta-based* consistency (Diskin, Xiong, Czarnecki, Ehrig, et al. 2011) and can be implemented by means of incremental model transformations (Kusel et al. 2013; Giese & Wagner 2008). It allows updating the model without overriding any changes. However, this requires fine-grained change sequences from the view (Kusel et al. 2013). Our work considers change sequences as a sequence of any possible single deterministic change to an EMOF-conforming model. As views are models themselves, this definition holds for both views and models.

In general, there are different types of views (Brunelière et al. 2018). In this paper, we focus on *projective* views which rely on an underlying model. In contrast, *synthetic* views form the model by their composition, i.e. the views themselves are the source of truth. Orthogonal to these types, *state-based views* are views that do not provide change sequences upon modification but instead only provide the modified view state. These are, for example, views that encapsulate artifacts of existing software applications to bridge between established software applications and processes. These existing applications are often not designed in a model-driven way and thus do not provide the required fine-grained change information when the view is modified (Sağlam & Kühn 2021). Furthermore, persistent, non-volatile unique element identifiers cannot either be assumed – due to the existing application or the view type definition — which makes identifying which elements of the original state changed significantly more complex.

Fortunately, model differencing (Kolovos et al. 2009; Stephan & Cordy 2013) can be employed to remedy this gap (Tunjic & Atkinson 2015). When utilizing model differencing to compare two states of the same model, i.e., the original state and a modified one, the derived changes describe a sequence of changes. This sequence can be considered as correct if the original model is transformed into the modified one by applying the sequence of changes (Cicchetti et al. 2007). *Similarity-based* (Lin et al. 2007; Treude et al. 2007; Toulmé & Inc 2006) model differencing approaches can be applied to any models while *identity-based* (Alanen & Porres 2003; Farail et al. 2006; Toulmé & Inc 2006) approaches are optimized for models with persistent unique identifiers (Brun & Pierantonio 2008). However, these model differencing approaches are not intended to operate with projections on models such as views. For model differencing, the compared models are assumed to contain all relevant information, and thus the correctness of the derived change sequence is judged based solely on them (Cicchetti et al. 2007). Views, in contrast, may represent only parts of the entire system. Therefore, when leveraging model differencing to compare view states for model-view consistency, the assumption that all relevant information is present does not hold anymore. Consequentially, the correctness criterion changes. Though changes are derived from the view states, their correctness must be determined in the context of the resulting state of the underlying model instead of the view, as it is the only source of *all* information. In summary, model differencing approaches were not designed to derive changes from view states to restore model-view consistency. However, they are required in order to support state-based views. Consequently, it is unclear how well current model differencing approaches are suitable for the consistency preservation of state-based views. Moreover, it is unclear how correctness can be defined concerning the underlying model for change sequences derived from view states.

This paper investigates the notion of correctness for change sequences in the context of model-view consistency. We also aim to understand how model differencing performs when deriving differences between projective view states to restore consistency with the underlying model. Thus, we make the following contributions:

Notion of Change Sequences Correctness (C1): We propose three properties of change sequences relevant for model-view consistency of projective views, which serve as different notions of correctness depending on the consistency constraint.

Evaluation of Model Differencing (C2): We evaluate how well EMF Compare (Eclipse Foundation 2019), as state-of-the-art model differencing tool, performs for model-view consistency. We conduct a case study with realistic evolution scenarios from the literature. We identify which cases produce incorrect results concerning each property. This indicates the relevance of our properties for model-view consistency.

Our results indicate that, in most cases, model differencing performs very well. However, incorrect change sequences were derived for two different common kinds of refactoring operations, which led to an incorrect model state. These types can be easily reproduced and are likely to occur in practice. We motivate how our introduced notions in combination with extensions to model differencing could have been used to prevent these incorrect results. In conclusion, our notion of change sequence correctness provides an improved understanding of model differencing in the context of model-view consistency.

This paper is structured as follows: In section 2, we introduce our running example of a contact tracing system to highlight challenges arising from model-driven view-based development, e.g., the model-view consistency problem. Then we define our notion of correctness for change sequences in section 3. Afterward, in section 4, we outline the case study we conducted to evaluate the relevance and significance of our properties, using a UML class model and Java code as views and EMF Compare (Eclipse Foundation 2019) to derive changes. In section 5, we discuss threats to the validity of our evaluation, and in section 6, related works in the field of view-based and model-driven development. Finally, section 7 concludes the paper.

2. Running Example

To illustrate the described challenges of view-based model-driven development, Figure 1 shows an example of a contact tracing system for disease control loosely based on the Covid-19 exposure notification APIs for mobile operating systems (Apple Inc. 2022; Google LLC 2022). The system tracks exposure to potentially infected users by anonymously recording encounters using a mobile application. When a person decides to test themselves for the disease, the testing laboratory notifies the health authority about the test. The health authority creates a report in a pending state for the respective user (`reports`). Any user having an encounter with the affected user within a specific time frame is added as an exposed user (`exposedTo`). The laboratory updates the report with the corresponding result when the test results are available. Lastly, users can monitor their risk status, which is determined based on their exposure.

The example encompasses three actors, namely users, laboratories, and the health authority. As the recorded data consists of sensitive geolocation and health data, every actor should only have access to a specific, limited amount of data. Furthermore, health domain logic needs to be abstracted away for users to understand their risk status. To fulfill these requirements, views can be utilized. Every actor has access to one or multiple views aiming at one particular purpose. For instance, the view for the risk status of a user computes an easily understandable risk value by aggregating all reports from users that the user was exposed to. Other views are the list of own anonymous encounters of individual users or an overview of all encounters for the health authority to populate reports with exposed users.

Next, we focus on the view of reports for the laboratory, depicted in Figure 1b. Every laboratory has a list of its diagnoses, which corresponds internally to the reports with the laboratory's identifier. Accordingly, the patient and diagnosis results correspond to the affected user, and the report result in the model. As the laboratory does not need any knowledge about exposed users, this information is not visible in the view. The view is integrated with the existing laboratory workflow, which does not monitor changes but only provides the modified state. To propagate changes from the state-based view back to the model, the necessary change sequence is derived using model differencing.

To highlight possible challenges of model differencing for model-view consistency, we consider the following modifications to the laboratory view. An employee of the laboratory enters the latest test results into the system. To increase efficiency, multiple results are entered at once. For every test, the `result` of the corresponding `Diagnosis` element is updated, and the `resultDate` is initialized to the current date. Finally, the employee confirms the changes, thus triggering the propagation from the view to the model. As part of this propagation, the difference between the initial and modified laboratory view states is determined via model differencing. Within this process, matching elements between the two states are collected. For each identified match, the (potential) difference between the two states of the element is determined and expressed in the change sequence. If no match is found, creation, property initialization changes, and respectively deletion changes — depending on whether the unmatched element is contained in the modified or initial state — are derived. In the laboratory view example, multiple elements remain identical in the modified state and are thus easy to match. These are all patients and all diagnoses that either got already evaluated or are still being processed. However, the modified diagnoses challenge model differencing as only some properties remain unchanged. More precisely, two out of four properties did change. Depending on the applied model differencing algorithm, this might result in no or incorrect matches for some or all modified elements. Consequently, in the absence of a match, the previous diagnosis element is deleted, and a new diagnosis is created. For incorrect matches, the elements are modified to match the element state of the modified view state. Depending on the degree of incorrect match, this might even include changing the `patient` of a diagnosis.

While in conventional model differencing scenarios, such deviating change sequences still result in the correct model state, although different operations were performed for model-view consistency, these deviations can have significant consequences. Propagating a change sequence that resulted from an unmatched diagnosis element — thus containing a deletion of the original diagnosis and a recreation of the modified one — triggers the deletion of the corresponding `Report` element in the model and the recreation of such with the modified values. As a result, the exposed users of the report are lost. This is caused by the absence of this information in the change sequence as the view does not represent this part of the model. Thus, the risk status of these users is not accurate anymore. For incorrectly matched diagnosis elements, it can even occur that reports are updated with wrong results leading again to an incorrect risk status.

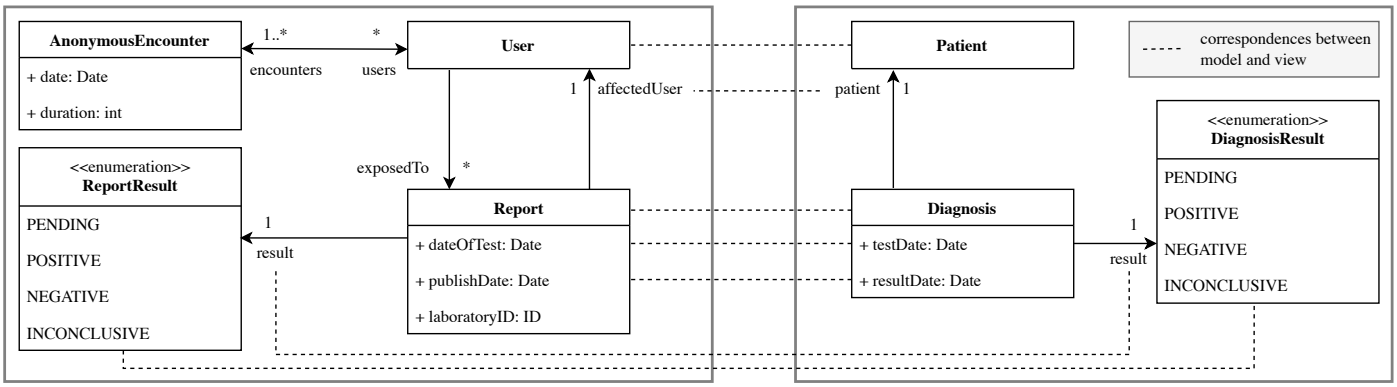
3. Change Sequence Properties

In this section, we propose the three properties *actual*, *admissible*, and *well-behaved* for change sequences that can serve as notions of correctness in the context of model-view consistency (C1) for projective views. The set relations of the three properties *actual*, *admissible*, and *well-behaved* are visualized in the Venn diagram in Figure 2 with respect to one actual change sequence. Henceforth, each property is individually introduced and defined.

Commonly, the main goal of model differencing between an initial and changed state of a model is to derive a change sequence that is as close as possible — regarding some metric — to the changes performed by a stakeholder (Cicchetti et al. 2007). We denote a change sequence *actual* if it exactly matches those changes.

Definition 1 (Actual Change Sequence) Let M be a model. Let M' be its modified state. A change sequence $\Delta_{M,M'}$ is *actual*, if it represents exactly the sequence of atomic changes executed by the stakeholder on M to obtain M'.

While it is desirable to exactly match the actual change sequence, in some cases, even with perfect heuristics, such is not possible. One case might be an actual change sequence that contains redundant changes, i.e., changes that can be removed from the change sequence without affecting the resulting model. Since



(a) Entire System Metamodel **(b)** Laboratory Results View Type

Figure 1 UML class diagram of a Contact Tracing System showcasing the underlying model (a) and a laboratory view type (b).

a view is a model itself, the admissibility property also applies for change sequences derived between view states. In contrast to model differencing, where a change sequence is the primary artifact, for model-view consistency change sequences are only a secondary artifact required to propagate changes from a view to the model. After change propagation, the change sequence can be discarded. This implies that the correctness of the change propagation is not validated on the change sequence but rather on the model. To emphasize this shift of focus, we propose the notion of *admissibility* of change sequences, defined as follows:

Definition 2 (Admissible Change Sequence) Let V be a view obtained by instantiating the view type VT on the model M. Let V' be its changed version obtained by applying the actual change sequence $\Delta^*_{V,V'}$ on V. Let $\Delta_{V,V'}$ be a change sequence derived from V and V'. Let M' be the updated model obtained by propagating $\Delta^*_{V,V'}$ from V to M. Then $\Delta_{V,V'}$ is *admissible* with respect to M, if propagating $\Delta_{V,V'}$ from V to M produces M'.

We denote a change sequence admissible if the updated model obtained by propagating the sequence is equal to the model that would have been obtained by propagating the actual change sequence. By defining the property inductively over the actual change sequence, it becomes independent of the representation of the change sequence, the change propagation mechanism, as well as the model representation. To put it succinctly, the change sequence derived from a view must only ensure that the impact on the model reflects the actual change. As we focus on projective views, this definition implies that not only the model reflects the actual change but also any other view within the system, as a projective view is completely described by its underlying model.

Consequently, an actual change sequence is always admissible with respect to itself. However, we can easily show that not every admissible change sequence is also an actual change sequence. Considering the contact tracing system example, a user might locally create an interaction with a duration of two minutes. The interaction then continues beyond the initially estimated two minutes and is thus updated to ten minutes. These values are then propagated to the model. Here, the actual change sequence consists of the initial interaction creation with a two-minute duration and the subsequent duration modification to ten minutes:

$$\begin{aligned}\Delta^*_{V,V'} = [\, & A = new\ AnonymousEncounter(),\\ & A.date = now(),\\ & A.duration = 2, A.duration = 10\,]\end{aligned}$$

In consequence, the same model state could be obtained by only including the last change to the duration:

$$\begin{aligned}\Delta_{V,V'} = [\, & A = new\ AnonymousEncounter(),\\ & A.date = now(),\\ & A.duration = 10\,]\end{aligned}$$

This sequence would be an admissible change sequence but not the actual one. While in this first example, the change sequence is admissible with respect to any model, the admissibility can also be context-dependent. To show this, we consider the described problematic scenario of the laboratory view. Here, the matching might fail and thus delete and recreate a report element instead of modifying it. Consequently, all exposed users are lost. However, if for the specific report there are no exposed users, the change sequence is admissible as any other information is present in the view and thus restored by the change sequence even though the old element gets deleted. Therefore, the admissibility property can be considered as a more relaxed constraint for model-view consistency, compared to the common requirements in model differencing, while still guaranteeing model correctness. Depending on the context, this relaxation can scale from excluding redundant changes of the actual change sequence or reordering independent changes to even replacing a set of changes with a distinct set of changes.

Regarding the view obtained from the updated model, an admissible change sequence does not imply any constraints on it. By definition, admissibility does not guarantee to solve the view-update problem, i.e., that the view retrieved from the updated model is equal to the changed view. Although not explicitly defined, in practice one would intuitively expect this to hold for any admissible change sequence as a correctly updated model should project the correct view. While this is true for

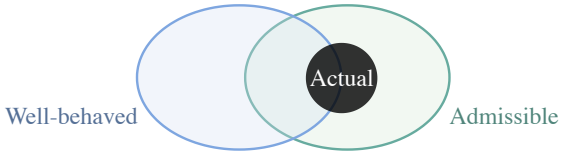


Figure 2 Venn Diagram outlining the relation between an actual change sequence and sets of admissible and well-behaved sequences.

the most common cases, there exist edge cases. As an example, in the contact tracing system all dates may get their seconds value trimmed to save storage space while in the laboratory view the seconds are preserved. In this scenario, regenerating the laboratory view after propagating an admissible change sequence will not produce an equal view as the date values are different. To formalize the expectation that a changed view remains the same after propagating the changes to the model, we propose the notion of *well-behavedness* of change sequences:

Definition 3 (Well-behaved Change Sequence) Let V be a view obtained by instantiating the view type VT on M. Let V' be its changed version. Let $\Delta_{V,V'}$ be a change sequence derived from V and V'. Let M' be the updated model obtained by propagating $\Delta_{V,V'}$ from V to M. Then $\Delta_{V,V'}$ is *well-behaved*, if V'' is obtained by instantiating VT on M' and $V'' = V'$.

A change sequence is well-behaved if the clean view state projected from the model after propagating the sequence is equal to the modified view state from which the sequence got derived. This definition follows the well-behavedness for (delta-)lenses, in particular its PUTGET law (Foster et al. 2005; Diskin, Xiong, & Czarnecki 2011). While an admissible change sequence does not constrain the view resulting from the updated model, a well-behaved change sequence does not constrain the updated model. However, in contrast to admissibility, it can be immediately observed whether a change sequence was well-behaved. This is a significant benefit as deviations from the expected state can directly be assessed by the stakeholder to potentially reject the performed changes.

As already motivated and shown in Figure 2, admissibility does not necessarily imply well-behavedness. Furthermore, there exist cases where a well-behaved change sequence is not admissible. An example for this is the scenario described for the laboratory view update in section 2. Here, changes to elements of the view (element `Diagnosis`) can impact elements that are not visible to the view (relation `exposedTo`). Without this property, all information could be regenerated from the information provided by the view, independent of the atomic changes of the derived change sequence. To generalize this, well-behavedness does not imply admissibility only in cases where changes to a view can possibly have an impact on elements that do not affect the view state.

In general, our new properties of well-behavedness and admissibility leverage the constraints of model differencing in model-view consistency. In particular, more change sequences than the actual change sequence can be assumed to produce correct results. Depending on the use case, the constraints on model-view consistency might be to simply preserve the changes performed on the view or to require all information in the model to be correctly updated. The former is fulfilled by well-behavedness, while the latter requires admissibility.

4. Case Study

The goals of our evaluation are twofold. First, we evaluate how state-of-the-art model differencing performs in common evolution scenarios in model-view consistency (C2). Second, we show that our new notions of change sequence correctness are relevant in practice. Thus, we want to assess whether our distinction of well-behaved, admissible, and actual change sequences occurs in realistic scenarios. Therefore, we developed a concept for a case study, selected relevant evolution scenarios, and implemented the concept for one particular domain pair to perform our evaluation on.

4.1. Method

As a framework for our evaluation, we constructed an exemplary case study (see Figure 3). Conceptually, we require a state-based view that leverages model differencing to derive and propagate changes. We call this view the *scenario view*. To obtain meaningful results, the underlying model must contain related information that is not present in the view. Otherwise, any well-behaved change sequence would automatically be admissible, as discussed in section 3. Mapping this to our running example (section 2), the scenario view would correspond to the laboratory results view, and the additional information missing in the view corresponds to the exposed users. To integrate this additional information into the model, an *additional view* is provided. We modeled this additional view to record its changes to avoid unintended interference by incorrect change sequences with our evaluation. However, the case study could also be instantiated with two state-based views, as this would simply increase its complexity. For every evolution scenario of the case study, the same test pipeline is executed:

1. the model is initially populated by propagating a modified scenario view,
2. the model is extended with the additional information absent in the scenario view by propagating a modified additional view,
3. a modified scenario view state is provided by each evolution scenario,
4. the change sequence between the scenario view state and the provided modified state is derived and propagated.

After the test of a scenario, the resulting model can be used to check the change sequence properties. The well-behavedness of the sequence is determined by comparing the scenario view projected from the resulting model with the modified view state provided by the evolution scenario. Its admissibility is determined by comparing the resulting model to a manually created expected model. Finally, the derived change sequence is compared to the actual change sequence. We require every actual

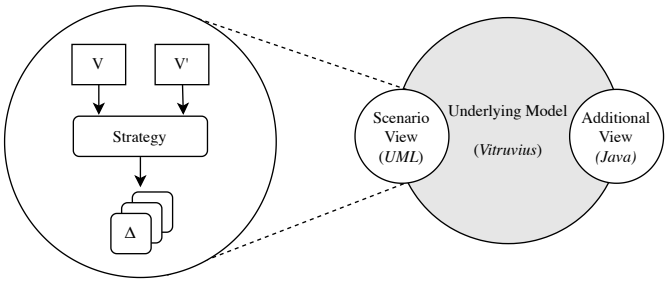


Figure 3 Overview of case study setup and its technical realization.

change sequence not to contain redundant changes, as the model differencing tools cannot derive any changes that are not visible in the modified view state. Additionally, we consider a change sequence as actual if it contains precisely the expected atomic changes independent of their order while still producing the expected resulting model. Since the order of changes to, for example, mutually independent properties is ambiguous, no heuristics exist to resolve such ambiguity and consequently, obtaining the correct order would require guessing. Thus, we opted for this relaxation to avoid misleading results. Notably, we do not check the change sequence derived in the first step of the test pipeline. This is not done as the model is initially empty. Thus, the derived change sequence is always expected to yield the correct model, following similar reasoning as for justifying the need for additional information. However, to avoid any setup mistakes, we validate the intermediate model state obtained after the initial two setup steps by comparing it to a manually created expected model.

4.2. Application Case

For the concrete case study, we chose the scenario view to model a UML class diagram and the additional view to represent Java source code. These domains were chosen as UML class diagrams are a standardized and widely used modeling approach, and Java is one of the most used programming languages in academia and industry (Ben Arfa Rabai et al. 2015). Furthermore, many semantic concepts are shared between the domains, but the Java source code contains some additional information, as required for our case study. In particular, we expect that for every UML package / class / operation / parameter / attribute there exists one equally named Java package / class / method / parameter / property with matching containment, and vice versa. Furthermore, element attributes like `final` or `static` as well as attribute and parameter types should be modeled consistently. In contrast, method bodies and property accessors are exclusive information of the Java domain. As a reference system, all scenarios are based on the `mediastore.basic` package of the Media Store case study (Strittmatter & Kechaou 2016), which we slightly adapted to support all of our evolution scenarios and to integrate with the considered UML and Java consistency relation. Its simplified UML class diagram is depicted in Figure 4. The diagram represents the expected state after the first two pipeline steps.

Regarding the test setup, the VITRUVIUS framework (Klare et al. 2021) is used as the foundation for the model-view consistency. VITRUVIUS is a framework to preserve consistency between individual models by using *Consistency Preservation Rules* and supports the projection of views. VITRUVIUS already provides a consistency specification for the UML and Java domains satisfying our described constraints which was thus used for our evaluation. The provided consistency specification does not perform any input-normalization (like the date normalization in the running example, see section 3). Consequently, we expect every admissible change sequence to be well-behaved. While we could have modified the consistency specification to artificially create a scenario for an admissible, non-well-behaved change sequence, we opted against it to preserve a realistic environment instead of covering every edge case. Still, we maintain that such edge cases might still exist in other domains and for other evolution scenarios.

For model differencing, we use the state-of-the-art framework *EMF Compare* (Eclipse Foundation 2019). EMF Compare is a model comparison framework that supports comparing and differencing any EMF-based models (Steinberg et al. 2009). By default, EMF Compare supports identity- and similarity-based matching when comparing models. The identity-based matching uses the element's identifier (e.g. for UML the XMI identifier), the similarity-based matching uses a weighted combination of metrics regarding the element's name, its type, its content, and its relation with other elements (Brun & Pierantonio 2008). While identity-based matching should be favored whenever possible due to its high precision and fast performance (Kolovos et al. 2009), persistent unique identifiers cannot be guaranteed when working with state-based views. Thus, we performed our evaluation with both the default identity- and similarity-based matching strategies of EMF Compare.

4.3. Evolution Scenarios

To obtain generalizable evolution scenarios relevant in practice, manually created models and change sequences based on realistic refactoring operations were used. This was mainly done due to two reasons. First, in preliminary tests, we detected that especially combined modifications to elements are challenging to the model differencing framework. Second, refactoring operations are well documented and already shown to occur frequently in real-world scenarios (Sidhu et al. 2018; Tsantalis et al. 2018).

Table 1 lists the considered evolution scenarios. Each scenario is based on the equally named refactoring operation as described by Sidhu et al. (2018). While these refactoring operations were explicitly created for UML class diagrams, the kinds of change can be mapped to other domains. Therefore, Table 1 additionally lists how the evolution scenarios relate to Java code refactorings (Tsantalis et al. 2013), model-to-model transformation refactorings (Wimmer et al. 2012), and generic code refactorings (Fowler 2019). Selecting evolution scenarios known to occur in various domains allows us to generalize our evaluation results beyond the scope of the two modeled domains.

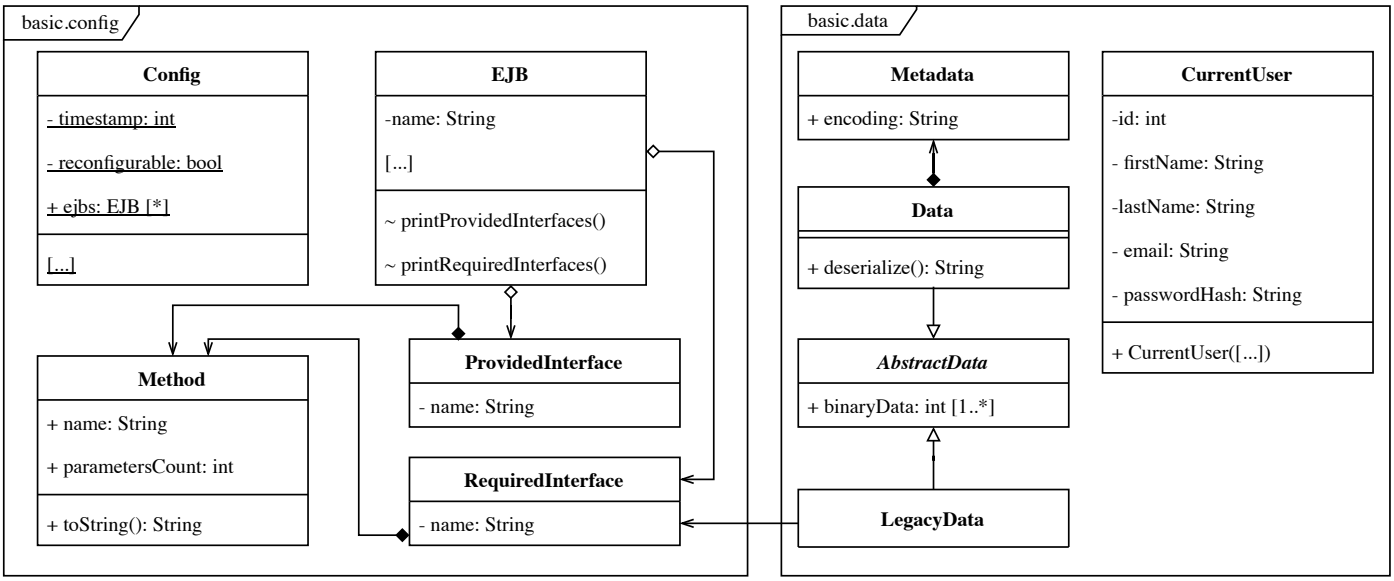


Figure 4 Initial state of the UML view (*scenario view*) in the case study of the Java and UML domains.

ID	Scenario (based on Sidhu et al. (2018))	Tsantalis et al. (2013)	Wimmer et al. (2012)	Fowler (2019)
CMS-	Change method signature - minor	-	1, 2	Change Function Declaration
CMS+	Change method signature - major	-	1, 2	Change Function Declaration
CH	Collapse hierarchy	Move Method, Move Field	5, 6, 8	Collapse Hierarchy
EAC	Extract associated class	Extract Method, Move Field	4, 7, 9	Extract Class
ES	Extract superclass	Extract Superclass	11	Extract Superclass
IC	Inline class	Move Method, Move Field	14, 15, 16	Inline Class
MC	Move class	Move Class	-	-
RAC	Remove associated class	-	-	-
RC-	Rename class - minor	Rename Class	3	-
RC+	Rename class - major	Rename Class	3	-

Table 1 Evolution scenarios used in our evaluation and their corresponding literature references.

Considering the evolution scenarios themselves, we want to exemplary describe the two *change method signature* (CMS-, CMS+) scenarios. Here, the method signature of `deserialize` of class `Data` is modified. This method is of particular interest because it has a method body in the Java view, which is unavailable in the UML view. The particular changes are making the method `static`, adding a new parameter named *data* of type `Data`, and changing the method name to *deserializeData* respectively *decodeData* for CMS- and CMS+. The expected result is that the method signature is correctly updated in both views and that the additional information (the method body) of the Java view is preserved.

4.4. Results

The results of our evaluation are summarized in Table 2. The identity-based strategy can derive the actual change sequence for every evolution scenario. Hence, every derived change sequence is, by definition, also admissible and well-behaved due to our change specifications. These results are anticipated, as the most complex task of model differencing — matching of elements — becomes trivial when identifiers are utilized.

In contrast, the similarity-based strategy cannot derive the actual change sequence for any scenario. By assessing the derived change sequences, one can observe that the similarity-based strategy has problems correctly matching elements with very little information. In particular, for the UML view, these are descendants of UML associations like `EAnnotation` of which multiple elements with identical properties can exist which are only differentiated by their containing element. Since the matching fails, the old elements are deleted, and new ones are created, resulting in additional changes which are not present in the actual change sequence. Nevertheless, the similarity-based strategy still derives in eight of ten cases an admissible change sequence. This observation emphasizes that it might be typical

	identity-based			similarity-based		
ID	**wbv**	**adm**	**act**	**wbv**	**adm**	**act**
CMS-	✓	✓	✓	✓	✓	×
CMS+	✓	✓	✓	✓	×	×
CH	✓	✓	✓	✓	✓	×
EAC	✓	✓	✓	✓	✓	×
ES	✓	✓	✓	✓	✓	×
IC	✓	✓	✓	✓	✓	×
MC	✓	✓	✓	✓	✓	×
RAC	✓	✓	✓	✓	✓	×
RC-	✓	✓	✓	✓	✓	×
RC+	✓	✓	✓	✓	×	×

wbv = well-behaved adm = admissible act = actual

Table 2 Change sequence properties per evolution scenario.

for a change sequence not to be actual but still result in the correct model state. In our case study, this is the case because there is no additional information linked to an `EAnnotation` besides the information present in the view. As such, the incorrectly regenerated elements recreate any information associated with the incorrectly deleted old elements. In both two scenarios where the change sequence is not admissible (CMS+, RC+), the underlying problem is that the strategy cannot identify a match for the renamed element. As such — similar to the `EAnnotation` problem — the unmatched elements are deleted and replaced with new instances. In these two scenarios, however, the incorrectly deleted elements (`Class` and `Method`) are linked to additional information not visible in the UML view, which is lost when recreating the elements. It is worth noting that the distance — with respect to some metric — between the names is critical as the CMS- and RC- scenarios, where the elements are renamed to a *more similar* name, produce admissible change sequences. Still, the similarity-based strategy can derive a well-behaved change sequence for every evolution scenario.

4.5. Discussion

From our results, we conclude that our distinction between actual, admissible, and well-behaved change sequences occurs in realistic scenarios with state-of-the-art model differencing tooling. Consequently, when designing view types and modifying state-based views, the required constraints on the model view consistency should be considered, which can be expressed in terms of our properties. If it is sufficient that all changes are correctly reflected in the view, i.e. that the view obtained from the updated model is identical to the view used to update the model, a well-behaved change sequence suffices. Exemplary scenarios would be the Java view in our case study — as all information of the UML view can be regenerated from Java information — or a system where all except one view are read-only. As in such a system only the one modifiable view can provide information to the underlying model, there cannot exist additional information besides those present in the considered view. As the well-behavedness of a change sequence can be automatically validated, any state-based view with this consistency constraint can be integrated into a delta-based consistency workflow without the danger of accidentally losing data by propagating an incorrectly derived change sequence. A simple implementation of such an integration would be to discard the transaction of propagating changes when it is detected that the change sequence is not well-behaved.

Requirement	**well-behaved**	**not well-behaved**
admissible	rejection	no feedback
not admissible	validation	*no purpose*

Table 3 Possibilities to automatically validate derived change sequences based on required properties imposed by the view.

However, requiring only a well-behaved change sequence might only be possible in some scenarios. For other, possibly more complex, scenarios, like the UML view in our case study, an admissible change sequence is required, as otherwise the information in the model hidden in the view is lost. Since an admissible change sequence requires the correctness of an *unknown* internal model state, it can never be automatically validated. Nevertheless, depending on the underlying model and the view, the admissibility of a change sequence might imply it being well-behaved. In such a case, any change sequences that are not well-behaved and thus also not admissible can automatically be rejected. Based on our findings, we expect admissible implying well-behaved to be the most common case.

For scenarios where the developer detects that an admissible and not necessarily well-behaved change sequence is required, a fully automated approach for any domain guaranteeing correctness is not possible. From our results, we see that identity-based matching has such high precision that one might assume that the actual change sequence is always derived from such views. However, at least for views using a similarity-based strategy, the constraint of deriving an admissible change sequence remains a challenge. An overview of the degree of automatic validation possible for given requirements on a derived change sequence is presented in Table 3. We consider views that neither require the change sequence to be admissible nor well-behaved to have no purpose, as this would imply that the derived change sequence can modify the model in any way without rejection, e.g. simply deleting all information.

In cases where a fully automatic validation is not possible, we envision, as a step towards a semi-automatic change sequence validation, the pre-processing of elements to filter out elements where the strategy is confident that it is a match. For the remaining elements, the user has to manually annotate matching elements, following the approach from Müller & Rumpe (2014). Another strategy to increase the degree of automation might be to annotate elements at model or meta-model level using domain knowledge, such that certain editing operations are forbidden.

Considering the incorrect update in our running example (section 2) where exposed users are lost due to incorrectly matched `Diagnosis` elements, annotating the view to require an admissible change sequence could have prevented this. In this case, the strategy could adapt to the increased consistency constraints, still identify matches for all unmodified elements, and remain the user with the task to match the few modified `Diagnosis`. As the domain prohibits the deletion of `Diagnosis` elements entirely, annotating the meta-model with this information could even allow to automatically reject certain incorrect change sequences.

5. Threats to Validity

After highlighting our findings, we discuss how we addressed threats to validity. We follow the guidelines of Wohlin et al. (2012) and Runeson & Höst (2008).

5.1. Internal Validity

Regarding internal validity, there is the risk that the observed failures result from an erroneous propagation of the changes to the model and not from incorrectly derived change sequences. However, we argue that this is not the case in our evaluation. We can reproduce the incorrect change sequences by using model differencing solely on the isolated view states, meaning without the propagation of changes from view to the model. Moreover, while employing specialized heuristics as pre-processing steps for the propagation of changes might help deal with faulty change sequences, these heuristics need to be domain-specific and designed for each view of any system. Consequently, we argue that there cannot be a domain-independent solution for model views. We maintain that the observed failures are not caused by the specific implementation of the used model differencing tool but rather conceptual issues of syntactic model differencing itself, as EMF Compare is widely used and heavily tested. Additionally, the execution of our evaluation is deterministic and independent of any human factors, as it is based on fully automated test cases.

5.2. External Validity

Regarding the external validity of our evaluation results, we argue that similar results can be obtained with other models of a different domain, as the problems shown in the change sequence are metamodel-independent. Furthermore, we argue that the evolution scenarios in our evaluation are relevant in practice, as they are taken from existing literature regarding common refactoring patterns for object-oriented software and models (Wimmer et al. 2012; Sidhu et al. 2018; Tsantalis et al. 2013; Fowler 2019). While it would have been possible to employ randomly generated evolution scenarios instead, most of the random scenarios could not be considered realistic. Thus, we opted for a realistic and well-established set of evaluation scenarios instead of a vast quantity of generated evaluation scenarios. Although we concede that our evaluation only employs EMF Compare, we argue that it is representative of syntactic model differencing, as it is the state-of-the-art and de facto standard tool. Other approaches also produce change sequences that describe the difference between model states and thus are arguably prone to the very same issues discussed in this paper.

6. Related Work

To our knowledge, no other work investigates the correctness of derived change sequences when employing model differencing for the model-view consistency of state-based views.

6.1. Model Differencing

Model Differencing has many applications, one of the most common being model versioning. Our work investigates model differencing to enable model-view consistency for state-based views. We focus on metamodel-independent approaches as view-based approaches need to be able to deal with multiple different metamodels. While model differencing has been widely researched (Stephan & Cordy 2013; Kolovos et al. 2009), most approaches are not designed for model views. One of the early approaches is *UMLDiff* (Xing & Stroulia 2005), which provides model differencing based on custom similarity metrics. Since these metrics are specifically designed and optimized for UML models, their application is limited to the UML domain. *DSMDiff* proposed a similar approach which rather supports arbitrary metamodels (Lin et al. 2007).

The most widely used metamodel-independent approach and thus the de-facto standard is *EMF Compare* (Brun & Pierantonio 2008). It uses similarity metrics for model matching and provides a high degree of customizability, thus allowing the design of custom strategies. As EMF Compare itself is model-driven, the derived changes are represented by a metamodel which enables further use of model transformations. In this paper, we thus base our evaluation on EMF Compare. Müller & Rumpe (2014) extend the EMF Compare framework by allowing to provide presettings that specify how certain elements have changed. While this can resolve any incorrect matches, this requires the knowledge of which matches need to be manually corrected, and thus additional effort is required. Metamodel-agnostic model differencing suffers from reduced accuracy due to the task's inherent complexity, which has been shown for EMF Compare and DSMDiff by Kolovos et al. (2009). Pietsch et al. (2013) present a set of model evolution scenarios that often pose problems for model differencing approaches. We modeled these scenarios as another case study for our evaluation but did not discuss them in detail as an admissible change sequence was always produced, independent of the matching strategy.

In contrast to the syntactic approaches like EMF Compare, which compares the concrete or abstract syntax of models, semantic model differencing compares models in terms of their meaning by leveraging semantic matching (Maoz et al. 2011; Maoz & Ringert 2016). Multiple approaches for semantic model differencing have been proposed (Langer et al. 2014; Addazi et al. 2016). Maoz & Ringert (2016) provide a framework to relate syntactic and semantic model differences. In our work, however, we focus on syntactic model differencing.

6.2. Model-View Approaches

View-based development allows to separate concerns and achieves to provide only the required information to different stakeholders (Cicchetti et al. 2019). The ISO standard 42010 (ISO/IEC/IEEE 42010:2011(E) 2011) provides a distinction between *projective* architectural views, where the views are projected from an underlying model, and *synthetic* architectural views, where a composition of views and their relations form the underlying model. This paper addresses projective views on models, as defined by Atkinson et al. (2010). Query-based views like *ModelJoin* (Burger et al. 2014), EMF views (Brunelière et al. 2015), and others (Werner et al. 2019) can be generated on the fly and their view types may change rapidly. Due to the scope of our work, we do not address these *flexible* (Burger 2013) views but focus on inflexible ones. Moreover, our work concerns *editable* (Atkinson et al. 2021) views.

There is a wide variety of many different model-view approaches (Brunelière et al. 2018) and multi-view modeling approaches (Cicchetti et al. 2019; Meier, Werner, et al. 2020). Meier, Kateule, & Winter (2020) introduce an approach to define viewpoints and views via reusable and generic operators, thus splitting the model-view transformation into parts. Atkinson & Tunjic (2017) present a conceptual framework for multi-dimensional views in projective modeling based on deep modeling. We base our evaluation on VITRUVIUS (Klare et al. 2021), which is one of the latter. VITRUVIUS is based on networks of transformations (Stevens 2020; Martínez et al. 2017; Gleitze et al. 2021; Klare et al. 2020; Sağlam & Klare 2021), where multiple binary model transformations form a network, thus keeping all models in the network consistent in a transitive manner. Alternatively, multiary transformations (Cleve et al. 2019; Bergmann 2021) can be employed instead.

6.3. View-Update Problem

While the view-update problem originates from database engineering (Bancilhon & Spyratos 1981), Foster et al. (2005, 2007) introduce an application of the view-update problem to model views based on the *lenses* framework. We relate to lenses, as our property well-behavedness of change sequences is equivalent to the PUTGET law for model transformations. However, we do not require the GETPUT law, as the former is sufficient in our case. Furthermore, our property admissibility is less restrictive regarding the view transformation than well-behaving lenses. Views can be kept consistent with their underlying models through model transformations (Foster et al. 2007). To this end, one can employ either general-purpose transformation languages or dedicated view transformation languages (Marussy et al. 2018).

While state-based approaches estimate the modifications between two model states, delta-based (Diskin, Xiong, Czarnecki, Ehrig, et al. 2011) approaches inductively define consistency preservation and thus avoid regenerating one of the models if the other is modified, which can lead to information loss due to overwriting. Delta-lenses (Diskin et al. 2010; Diskin, Xiong, Czarnecki, Ehrig, et al. 2011) or incremental transformations (Giese & Wagner 2006, 2008; Leblebici et al. 2017; Beaudoux et al. 2010) can be employed for delta-based model-view consistency. Some of the most notable works regarding incremental model-view consistency are discussed in the following. Semeráth et al. (2016) introduce an approach to generate delta-based backward transformations when creating views from models via unidirectional forward transformations. Debreceni et al. (2014) leverage declarative model queries to define views automatically. The views' existence is entirely bound to the underlying model. Thus views are automatically and incrementally maintained. Marussy et al. (2018) propose an incremental view model synchronization approach that provides a fully compositional transformation language. It is based on delta-based views and tolerates inconsistencies, thus allowing for partial consistency. Furthermore, numerous approaches allow delta-based consistency preservation for multiple models, as shown by Stünkel et al. (2021). Examples include cross-domain consistency checking (Torres et al. 2021) and model repair (Macedo et al. 2017). Some approaches, such as many SUM-based approaches (Meier, Werner, et al. 2020), specifically include support for projective views. However, all aforementioned delta-based approaches require views to provide fine-grained changes upon modification (Kusel et al. 2013), for example, through an observer that tracks model changes. As such, our work can be considered an extension to such approaches to bridge between delta-based change propagation and state-based views.

6.4. State-based Views

State-based views are views that, as opposed to delta-based views, do not provide this fine-grained change sequence upon modification but instead only provide the modified view state. This is often the case when using existing tooling for views (Sağlam & Kühn 2021). To close this gap, we investigate model differencing to provide these change sequences, thus allowing to leverage delta-based consistency preservation for state-based views. The previously mentioned approaches require delta-based views. However, we show that fine-grained changes cannot always be derived. As such, we investigate the correctness of derived change sequences when employing model differencing for state-based views. We thus relate to Fritsche et al. (2018), which study how to cope with change sequences for models that delete and recreate elements instead of updating them. They mitigate the issue of potentially losing information by introducing *short-cut operations*, which transform change sequences to update the elements instead. In some cases, which we observed in our case study, short-cut operations could be leveraged as customization to EMF Compare to reduce the incorrectness of the change sequences. Tunjic & Atkinson (2015) introduce an approach for simplifying view type definition, using the lenses framework for model-view consistency. Moreover, they propose using model differencing to derive changes between view states as part of their approach. However, to avoid the view-update problem entirely, they restrict the view construction mechanism and thus reduce the expressiveness of the possible views. Our work has no restrictions to the view construction or view expressiveness. Their work focuses on the change propagation into the model, thus assuming the actual change sequence is given. They do not specify constraints on model differencing or the derived change sequence.

7. Conclusion

View-based model-driven development approaches need to ensure the consistency of the projective views with their underlying model. As one approach to model-view consistency, delta-based consistency preservation requires views to provide fine-grained changes upon modifications (Kusel et al. 2013). Yet, state-based views do not provide such a change sequence but only the modified view state. To close this gap, model differencing can be employed (Tunjic & Atkinson 2015) to derive change sequences from view states. However, model differencing approaches were not intended to operate with views on models. Thus, it needed to be clarified how well-suited a state-of-the-art model differencing approach is for state-based views. To this end, this paper makes two contributions. First (C1), we propose three properties of change sequences for model-view consistency, i.e., *actual*, *admissible*, and *well-behaved*, to distinguish different notions of correctness depending on the consistency constraints. While actual change sequences match the exact changes of the stakeholder, admissible change sequences guarantee that the model is correctly updated, whereas well-behaved change sequences ensure that the view is identical to a newly instantiated view type from the updated model. As actual and admissible change sequences rely on hidden information, only well-behaved change sequences can automatically be validated. Second (C2), we evaluated how well EMF Compare — as a state-of-the-art model differencing tool — performs when employed for model-view consistency. We devised a realistic case study with two views, which we edited based on realistic evolution scenarios retrieved from the literature. Our results identified two typical evaluation scenarios for which EMF Compare produces incorrect results when using a similarity-based strategy.

In conclusion, our study supports that our distinction between actual, admissible, and well-behaved change sequences occurs in realistic scenarios. While the tested identity-based strategy is always able to derive the actual change sequence, the similarity-based strategy never derives it. However, an admissible change sequence is derived in most cases, and a well-behaved one is always derived. Based on these findings, a view requiring admissible change sequences can be integrated into a delta-based consistency workflow with minimal risk if it supports identity-based matching. If only similarity-based matching is available, a risk-free integration is only possible if the view is annotated to require only a well-behaved change sequence. For any other case, we envision adaptions to existing model differencing approaches to support semi-automated change derivation to avoid accidental data loss. In future work, we plan to study projects with available version histories to inspect differences between revisions and to evaluate further model differencing tools besides EMF Compare. Additionally, we want to investigate the impact of our properties in scenarios with concurrent edits of different views. In sum, we conclude that in order to support state-based views in view-based model-driven development, state-of-the-art model differencing approaches are well-suited. However, in cases where identity-based matching is impossible, depending on the required change sequence property, semi-automated change derivation should be employed.

Verifiability

The implementation of VITRUVIUS (Klare et al. 2021) is available at GitHub (Vitruv Tools 2021). In addition, we provide a dedicated reproduction package (Wittler et al. 2023), which contains the necessary artifacts for reproducing our evaluation.

Acknowledgments

The authors would like to thank Heiko Klare for the insightful discussions and constructive criticism of the manuscript. This publication is partially based on the research project *SofDCar* (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action. It is also funded by the Baden-Württemberg Ministry of Science, Research and the Arts within the *InnovationCampus Future Mobility*.

References

Addazi, L., Cicchetti, A., Di Rocco, J., Di Ruscio, D., Iovino, L., & Pierantonio, A. (2016, 10). Semantic-based model matching with emfcompare. In T. Mayerhofer, A. Pierantonio, B. Schätz, & D. Tamzalit (Eds.), *10th workshop on models and evolution* (pp. 40–49). CEUR-WS. Retrieved from http://www.es.mdh.se/publications/4468-

Alanen, M., & Porres, I. (2003). Difference and union of models. In P. Stevens, J. Whittle, & G. Booch (Eds.), *«uml» 2003 - the unified modeling language. modeling languages and applications* (pp. 2–17). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-540-45221-8_2

Apple Inc. (2022). *Exposure Notification Documentation.* Retrieved from https://developer.apple.com/documentation/exposurenotification (Accessed: 2022-11-28)

Atkinson, C., Stoll, D., & Bostan, P. (2010). Orthographic Software Modeling: A Practical Approach to View-Based Development. In L. Maciaszek, C. González-Pérez, & S. Jablonski (Eds.), *Evaluation of novel approaches to software engineering* (Vol. 69, pp. 206–219). Berlin/Heidelberg: Springer. doi: 10.1007/978-3-642-14819-4_15

Atkinson, C., & Tunjic, C. (2017, oct). A deep view-point language for projective modeling. *Information Systems*, *101*, 101440. doi: 10.1109/edoc.2017.26

Atkinson, C., Tunjic, C., & Lange, A. (2021, oct). Controlling view editability in projection-based modeling environments. In *2021 IEEE 25th international enterprise distributed object computing conference (EDOC)* (pp. 152–161). Los Alamitos, CA, USA: IEEE. doi: 10.1109/edoc52215.2021.00026

Bancilhon, F., & Spyratos, N. (1981, December). Update semantics of relational views. *ACM Trans. Database Syst.*, *6*(4), 557–575. doi: 10.1145/319628.319634

Beaudoux, O., Blouin, A., Barais, O., & Jézéquel, J.-M. (2010). Active operations on collections. In D. C. Petriu, N. Rouquette, & Ø. Haugen (Eds.), *Model driven engineering languages and systems* (pp. 91–105). Berlin, Heidelberg: Springer Berlin Heidelberg.

Ben Arfa Rabai, L., Cohen, B., & Mili, A. (2015, 10). Programming language use in us academia and industry. *Informatics in Education*, *14*, 143–160. doi: 10.15388/infedu.2015.09

Bergmann, G. (2021, Oct 01). Controllable and decomposable multidirectional synchronizations. *Software and Systems Modeling*, *20*(5), 1735–1774. doi: 10.1007/s10270-021-00879-w

Brun, C., & Pierantonio, A. (2008). Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, *9*(2), 29–34.

Brunelière, H., Burger, E., Cabot, J., & Wimmer, M. (2018). A feature-based survey of model view approaches. In *Proceedings of the 21th acm/ieee international conference on model driven engineering languages and systems* (p. 211). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3239372.3242895

Brunelière, H., Perez, J. G., Wimmer, M., & Cabot, J. (2015, 10). EMF views: A view mechanism for integrating heterogeneous models. In P. Johannesson, M. L. Lee, S. W. Liddle, A. L. Opdahl, & Ó. Pastor López (Eds.), *34th International Conference on Conceptual Modeling (ER 2015)* (pp. 317–325). Cham: Springer International Publishing. doi: 10.1007/978-3-319-25264-3_23

Burger, E. (2013). Flexible Views for View-Based Model-Driven Development. In *Proceedings of the 18th international doctoral symposium on components and architecture* (pp. 25–30). New York, NY, USA: ACM. doi: 10.1145/2465498.2465501

Burger, E., Henß, J., Küster, M., Kruse, S., & Happe, L. (2014). View-Based Model-Driven Software Development with ModelJoin. *Software & Systems Modeling*, *15*(2), 472–496. doi: 10.1007/s10270-014-0413-5

Cicchetti, A., Ciccozzi, F., & Pierantonio, A. (2019, 12). Multi-view approaches for software and system modelling: a systematic literature review. *Software & Systems Modeling*, *18*, 3207–3233. doi: 10.1007/s10270-018-00713-w

Cicchetti, A., Di Ruscio, D., & Pierantonio, A. (2007, 10). A metamodel independent approach to difference representation. *Journal of Object Technology*, *6*, 165–185. doi: 10.5381/jot.2007.6.9.a9

Cleve, A., Kindler, E., Stevens, P., & Zaytsev, V. (2019). Multidirectional Transformations and Synchronisations (Dagstuhl Seminar 18491). *Dagstuhl Reports*, *8*(12), 1–48. doi: 10.4230/DagRep.8.12.1

Debreceni, C., Horváth, A., Hegedüs, A., Ujhelyi, Z., Ráth, I., & Varró, D. (2014). Query-driven incremental synchronization of view models. In *Proceedings of the 2nd workshop on view-based, aspect-oriented and orthographic software modelling* (p. 31–38). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/2631675.2631677

Diskin, Z., Xiong, Y., & Czarnecki, K. (2010). From state- to delta-based bidirectional model transformations. In L. Tratt & M. Gogolla (Eds.), *Theory and practice of model transformations* (pp. 61–76). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-13688-7_5

Diskin, Z., Xiong, Y., & Czarnecki, K. (2011). From state- to delta-based bidirectional model transformations: the asymmetric case. *Journal of Object Technology*, *10*, 6:1–25. doi: 10.5381/jot.2011.10.1.a6

Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., & Orejas, F. (2011). From state- to delta-based bidirectional model transformations: The symmetric case. In *Model driven engineering languages and systems* (Vol. 6981, pp. 304–318). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-24485-8_22

Eclipse Foundation. (2019). *EMF Compare.* https://www.eclipse.org/emf/compare. (Accessed: 2022-11-28)

Farail, P., Gaufillet, P., Canals, A., LE Camus, C., Sciamma, D., Michel, P., ... Pantel, M. (2006, January). The TOPCASED project: a Toolkit in Open source for Critical Aeronautic SystEms Design. In *Conference ERTS'06.* Toulouse, France. Retrieved from https://hal.archives-ouvertes.fr/hal-02270461

Foster, J. N., Greenwald, M. B., Moore, J. T., Pierce, B. C., & Schmitt, A. (2005, January). Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. *SIGPLAN Not.*, *40*(1), 233–246. doi: 10.1145/1047659.1040325

Foster, J. N., Greenwald, M. B., Moore, J. T., Pierce, B. C., & Schmitt, A. (2007, May). Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, *29*(3), 17–es. doi: 10.1145/1232420.1232424

Fowler, M. (2019). *Refactoring: Improving the design of existing code* (Second edition ed.). Boston, MA, USA: Addison-Wesley.

Fritsche, L., Kosiol, J., Schürr, A., & Taentzer, G. (2018). Short-cut rules. In M. Mazzara, I. Ober, & G. Salaün (Eds.), *Software technologies: Applications and foundations* (pp. 415–430). Cham: Springer International Publishing. doi: 10.1007/978-3-030-04771-9_30

Giese, H., & Wagner, R. (2006). Incremental model synchronization with triple graph grammars. In *Model driven engineering languages and systems* (pp. 543–557). Berlin, Heidelberg: Springer Berlin Heidelberg.

Giese, H., & Wagner, R. (2008, 3). From model transformation to incremental bidirectional model synchronization. *Software & Systems Modeling*, *8*(1), 21–43. doi: 10.1007/s10270-008-0089-9

VITRUVIUS *GitHub Organization.* (2021). https://github.com/vitruv-tools. (Accessed: 2022-11-28)

Gleitze, J., Klare, H., & Burger, E. (2021). Finding a universal execution strategy for model transformation networks. In *Fundamental approaches to software engineering* (pp. 87–107). Cham: Springer International Publishing. doi: 10.1007/978-3-030-71500-7_5

Goldschmidt, T., Becker, S., & Burger, E. (2012, March). Towards a tool-oriented taxonomy of view-based modelling. In E. J. Sinz & A. Schürr (Eds.), *Proceedings of the modellierung 2012* (Vol. P-201, pp. 59–74). Bonn, Germany: Gesellschaft für Informatik e.V. (GI).

Google LLC. (2022). *Exposure Notifications API.* Retrieved from https://developers.google.com/android/exposure-notifications/exposure-notifications-api (Accessed: 2022-11-28)

ISO/IEC/IEEE 42010:2011(E). (2011). *Systems and software engineering – architecture description*. International Organization for Standardization, Geneva, Switzerland. doi:

10.1109/IEEESTD.2011.6129467
Kent, S. (2002). Model driven engineering. , 286–298. doi: 10.1007/3-540-47884-1_16
Klare, H., Kramer, M. E., Langhammer, M., Werle, D., Burger, E., & Reussner, R. (2021). Enabling consistency in view-based system development — the vitruvius approach. *Journal of Systems and Software*, *171*, 110815. doi: 10.1016/j.jss.2020.110815
Klare, H., Pepin, A., Burger, E., & Reussner, R. (2020). *A Formal Approach to Prove Compatibility in Transformation Networks* (Vol. 2020; Tech. Rep. No. 3). Karlsruhe: Karlsruhe Institute of Technology (KIT). doi: 10.5445/IR/1000121444
Kolovos, D., Di Ruscio, D., Pierantonio, A., & Paige, R. (2009, 5). Different models for model matching: An analysis of approaches to support model differencing. In *2009 ICSE workshop on comparison and versioning of software models* (pp. 1–6). doi: 10.1109/CVSM.2009.5071714
Kusel, A., Etzlstorfer, J., Kapsammer, E., Langer, P., Retschitzegger, W., Schoenboeck, J., ... Wimmer, M. (2013). A survey on incremental model transformation approaches. In *Me 2013 – models and evolution workshop proceedings* (pp. 4–13).
Langer, P., Mayerhofer, T., & Kappel, G. (2014). Semantic model differencing utilizing behavioral semantics specifications. In J. Dingel, W. Schulte, I. Ramos, S. Abrahão, & E. Insfran (Eds.), *Model-driven engineering languages and systems* (pp. 116–132). Cham: Springer International Publishing. doi: 10.1007/978-3-319-11653-2_8
Leblebici, E., Anjorin, A., Fritsche, L., Varró, G., & Schürr, A. (2017). Leveraging incremental pattern matching techniques for model synchronisation. In J. de Lara & D. Plump (Eds.), *Graph transformation* (pp. 179–195). Cham: Springer International Publishing.
Lin, Y., Gray, J., & Jouault, F. (2007, 08). Dsmdiff: A differentiation tool for domain-specific models. *European Journal of Information Systems - EUR J INFOR SYST*, *16*, 349–361. doi: 10.1057/palgrave.ejis.3000685
Macedo, N., Jorge, T., & Cunha, A. (2017). A Feature-based Classification of Model Repair Approaches. *IEEE Transactions on Software Engineering*, *43*(7), 615–640. doi: 10.1109/TSE.2016.2620145
Maoz, S., & Ringert, J. O. (2016, aug). A framework for relating syntactic and semantic model differences. *Software and Systems Modeling*, *17*(3), 753–777. doi: 10.1007/s10270-016-0552-y
Maoz, S., Ringert, J. O., & Rumpe, B. (2011). A manifesto for semantic model differencing. In J. Dingel & A. Solberg (Eds.), *Models in software engineering* (pp. 194–203). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-21210-9_19
Martínez, S., Tisi, M., & Douence, R. (2017). Reactive model transformation with atl. *Science of Computer Programming*, *136*, 1–16. doi: 10.1016/j.scico.2016.08.006
Marussy, K., Semeráth, O., & Varró, D. (2018). Incremental view model synchronization using partial models. In *Proceedings of the 21th acm/ieee international conference on model driven engineering languages and systems* (p. 323–333). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/3239372.3239412
Meier, J., Kateule, R., & Winter, A. (2020). Operator-based viewpoint definition. In *Proceedings of the 8th international conference on model-driven engineering and software development - modelsward,* (pp. 401–408). SciTePress. doi: 10.5220/0008977404010408
Meier, J., Werner, C., Klare, H., Tunjic, C., Aßmann, U., Atkinson, C., ... Winter, A. (2020). Classifying approaches for constructing single underlying models. In *Model-driven engineering and software development* (pp. 350–375). Cham: Springer International Publishing. doi: 10.1007/978-3-030-37873-8_15
Murer, S., Bonati, B., & Furrer, F. J. (2011). *Managed evolution: A strategy for very large information systems* (1st ed.). Springer Berlin Heidelberg. doi: 10.1007/978-3-642-01633-2
Müller, K., & Rumpe, B. (2014, Feb). User-Driven Adaptation of Model Differencing Results. In *Software engineering 2014 : Fachtagung des gi-fachbereichs softwaretechnik ; 25. februar - 28. februar 2014 in kiel, deutschland / wilhelm hasselbring ... (hrsg.)* (pp. 25–29). Bonn: Ges. für Informatik. doi: 10.13140/2.1.2796.7682
Pietsch, P., Müller, K., & Rumpe, B. (2013). Model matching challenge: Benchmarks for ecore and bpmn diagrams. *Softwaretechnik-Trends*, *33*(2), 95–100. doi: 10.1007/s40568-013-0061-x
Runeson, P., & Höst, M. (2008, dec). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. doi: 10.1007/s10664-008-9102-8
Sağlam, T., & Klare, H. (2021). Classifying and avoiding compatibility issues in networks of bidirectional transformations. In *STAF 2021 workshop proceedings: 9th international workshop on bidirectional transformations.* CEUR-WS. (accepted, to appear)
Sağlam, T., & Kühn, T. (2021, oct). 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)* (p. 410-416). IEEE. doi: 10.1109/MODELS-C53483.2021.00064
Semeráth, O., Debreceni, C., Horváth, A., & Varró, D. (2016). Incremental backward change propagation of view models by logic solvers*. In *Proceedings of the acm/ieee 19th international conference on model driven engineering languages and systems* (p. 306–316). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/2976767.2976788
Sidhu, B. K., Singh, K., & Sharma, N. (2018). A catalogue of model smells and refactoring operations for object-oriented software. In *2018 second international conference on inventive communication and computational technologies (icicct)* (pp. 313–319). doi: 10.1109/ICICCT.2018.8473027
Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2009). *Emf: Eclipse modeling framework 2.0* (2nd ed.). Addison-Wesley Professional.
Stephan, M., & Cordy, J. R. (2013). A survey of model compar-

ison approaches and applications. In *Proceedings of the 1st international conference on model-driven engineering and software development - volume 1: Modelsward,* (pp. 265–277). SciTePress. doi: 10.5220/0004311102650277

Stevens, P. (2020). Maintaining consistency in networks of models: bidirectional transformations in the large. *Software and Systems Modeling*, *19*(1), 39–65. doi: 10.1007/s10270-019-00736-x

Strittmatter, M., & Kechaou, A. (2016). *The media store 3 case study system* (Vol. 2016; Tech. Rep. No. 1). Karlsruher Institut für Technologie (KIT). doi: 10.5445/IR/1000052197

Stünkel, P., König, H., Rutle, A., & Lamo, Y. (2021, January). Multi-model evolution through model repair. *Journal of Object Technology*, *20*(1), 1:1–25. (Workshop on Models and Evolution (ME 2020)) doi: 10.5381/jot.2021.20.1.a2

Torres, W., van den Brand, M. G. J., & Serebrenik, A. (2021, 10). A systematic literature review of cross-domain model consistency checking by model management tools. *Softw. Syst. Model.*, *20*(3), 897–916. doi: 10.1007/s10270-020-00834-1

Toulmé, A., & Inc, I. (2006). Presentation of emf compare utility. In *Eclipse modeling symposium* (Vol. 1).

Treude, C., Berlik, S., Wenzel, S., & Kelter, U. (2007). Difference computation of large models. In *Proceedings of the the 6th joint meeting of the european software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering - ESEC-FSE '07* (pp. 295–304). New York, NY, USA: ACM Press. doi: 10.1145/1287624.1287665

Tsantalis, N., Guana, V., Stroulia, E., & Hindle, A. (2013). A multidimensional empirical study on refactoring activity. In *Proceedings of the 2013 conference of the center for advanced studies on collaborative research* (p. 132–146). USA: IBM Corp.

Tsantalis, N., Mansouri, M., Eshkevari, L., Mazinanian, D., & Dig, D. (2018). Accurate and efficient refactoring detection in commit history. In *2018 ieee/acm 40th international conference on software engineering (icse)* (pp. 483–494). doi: 10.1145/3180155.3180206

Tunjic, C., & Atkinson, C. (2015). Synchronization of projective views on a single-underlying-model. In *Proceedings of the 2015 joint morse/vao workshop on model-driven robot software engineering and view-based software-engineering* (pp. 55–58). New York, NY, USA: ACM. doi: 10.1145/2802059.2802066

Werner, C., Wimmer, M., & Assmann, U. (2019, sep). A generic language for query and viewtype generation by-example. In *2019 ACM/IEEE 22nd international conference on model driven engineering languages and systems companion (MODELS-c)* (pp. 379–386). Los Alamitos, CA, USA: IEEE. doi: 10.1109/models-c.2019.00059

Wimmer, M., Martínez, S., Jouault, F., & Cabot, J. (2012, August). A catalogue of refactorings for model-to-model transformations. *Journal of Object Technology*, *11*(2), 2:1–40. doi: 10.5381/jot.2012.11.2.a2

Wittler, J. W., Sağlam, T., & Kühn, T. (2023, May). Replication package for the paper "evaluating model differencing for the consistency preservation of state-based views". doi: 10.5281/zenodo.7945324

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-29044-2

Xing, Z., & Stroulia, E. (2005). Umldiff: An algorithm for object-oriented design differencing. In *Proceedings of the 20th ieee/acm international conference on automated software engineering* (p. 54–65). New York, NY, USA: Association for Computing Machinery. doi: 10.1145/1101908.1101919

About the authors

Jan Willem Wittler is a doctoral researcher at the KASTEL – Institute of Information Security and Dependability at Karlsruhe Institute of Technology (KIT) since 2022. His research interests involve managing the evolution of variant-rich cyber-physical systems in a consistency-preserving manner and view-based modeling. You can contact the author at research@wittler.app or visit https://dsis.kastel.kit.edu/alumni_jan_wittler.php.

Timur Sağlam is a doctoral researcher at the KASTEL – Institute of Information Security and Dependability at Karlsruhe Institute of Technology (KIT) since 2020. His primary research interests involve software plagiarism and collusion detection, particularly obfuscation attacks on structure-based plagiarism detectors for code and modeling artifacts. Furthermore, he is interested in the consistency preservation of models and views, and the adoption of model-driven approaches and tools. You can contact the author at timur.saglam@kit.edu or visit https://dsis.kastel.kit.edu/staff_saglam.php.

Thomas Kühn is a postdoctoral researcher at the Software Engineering and Programming Languages group at Martin Luther University Halle-Wittenberg. He received his Ph.D. in 2017 at the Dresden University of Technology. His research focuses on new ways to model and program future software systems challenged by increased complexity, heterogeneity, rate of change, and longevity. As a result, he investigates how software product line engineering can be applied to systems engineering and software language engineering. You can contact the author at thomas.kuehn@informatik.uni-halle.de or visit https://swt.informatik.uni-halle.de/mitarbeiter/45865_3467283.