Towards the Co-Evolution of Models and Artefacts of Industrial Tools Through External Views

Timur Sağlam
KASTEL Institute of Information Security and Dependability
Karlsruhe Institute of Technology
Karlsruhe, Germany
timur.saglam@kit.edu

Thomas Kühn
KASTEL Institute of Information Security and Dependability
Karlsruhe Institute of Technology
Karlsruhe, Germany
thomas.kuehn@kit.edu

***Abstract*—Modern software systems comprise multiple models. When these models are changed, interdependent models must be evolved accordingly. Manually managing this co-evolution of models is tedious and error-prone. Moreover, other interdependent artefacts, such as persisted states of industrial software applications, must co-evolve accordingly. Automated consistency preservation allows for efficiently managing the co-evolution of models. However, while state-of-the-art approaches operate delta-based, typical software applications persist changes state-based without conforming to explicit metamodels. Additionally, software applications may persist changes infrequently, even though interdependent models might be concurrently modified. As such, current approaches are insufficient for artefacts of industrial tools. To address these issues, we propose an approach for the co-evolution of models and artefacts of industrial tools by treating these artefacts as external views on the models.**

***Index Terms*—Co-Evolution, Consistency Preservation, Model-View Consistency, View-Based Development, MDE Adoption**

I. Introduction

Since modern software systems can become very complex and large-scale, their development often involves multiple models. Especially in agile development, these models change often, and interdependent models must co-evolve accordingly. Manually managing this co-evolution of models is tedious and error-prone. Automated consistency preservation manages the co-evolution of models and thus addresses this problem [1], [2]. However, during the Co-Evolution of Models, we additionally need to evolve other interdependent artefacts [3]. Specifically, these artefacts could be informal models persisted by tools that do not operate in a model-driven way, such as industrial (or commercial) software applications. They cannot automatically evolve with models when they only have a limited interface for publishing changes such as state-based persistence through XML or JSON files. Often these artefacts do not conform to an explicit metamodel and hide the changes made by the applications. Even the persisted states only provide an abstraction of the actual fine-grained changes. As state-of-the-art consistency preservation approaches primarily employ delta-based changes, i.e., they continuously observe fine-grained changes to models and update the interdependent models incrementally, these approaches are not applicable for state-based artefacts of industrial tools. Vice versa, industrial tools are often legacy software, which cannot be easily adapted or replaced to be compatible with MDE approaches, such as model-based consistency preservation, as they might be closed-source or simply too complex. [4], [5]. Moreover, while consistency preservation allows for immediately synchronizing models upon change, industrial tools might synchronize their state with a considerably lower frequency. Often the tool's internal state is only persisted when the user explicitly decides to save their changes. This can be a matter of minutes, but also a matter of hours. Interdependent models, however, may change while the industrial tool is not synchronized. Consequently, conflicts may arise when the industrial tool's state is later synchronized.

To address this issue, we propose an approach for the co-evolution of models and artefacts of industrial tools. We treat these artefacts as views on a model in the system, thus breaking down the problem into two parts: First, the model-view consistency between the industrial tool and the interdependent model. Second, the co-evolution of all models in the system. While the latter can be realized using existing consistency preservation approaches, the former needs to be supported by a definition of *external views*. An external view is a view that encapsulates a black-box software system that reads and persists its state in a specific format. The external view acts as an adapter between the modeling system and the industrial tools, thus bridging the gap between them. We aim to enable the automated co-evolution of models and artefacts of industrial tools. By leveraging external views, we want to develop a partly automated approach independent of the specific models or the industrial tools. Thus, we make the following contributions with this paper:

Co-Evolution Approach (C1): We propose an approach for the co-evolution of artefacts of tools with models leveraging consistency preservation and external views.

Key Challenges (C2): We identify key challenges for the co-evolution of artefacts of industrial tools with multiple models.

II. State of the Art and Related Work

To relate our approach to the state-of-the-art, we henceforth discuss relevant research areas.

A. View-Based Development

We base our approach on model *views*. Views enable users to see a system from a specific viewpoint. While a view might represent (*show*) a model, it can be considered as a model itself. However, a view is dynamically generated [6]. A *view type* is a metamodel for a set of views and describes the contained elements and relations [7]. We distinguish between pre-defined views and *flexible* views, which can be defined on the fly to accustom individual needs of a user [8]. While some approaches materialize views as copies of the represented models, others are based on virtual models [9].

The problem of updating the model of a view when the view is changed is called the *view-update problem*. While this problem originates from database engineering [10], Foster, Greenwald, Moore, *et al.* [11], [12] introduce an application of the view-update problem to model views. Views can be kept consistent with their underlying models through *bidirectional transformations* (BX), as proposed by the lenses framework [12]. Delta-lenses [13] or incremental transformations [14], [15] can be employed for model-view consistency to avoid regenerating one of the other is changed. Finally, views can be defined for modeling systems, for example, with single-underlying models (SUMs) [16] or megamodels [17]. There is a wide variety of model-view approaches [18] and multi-view modeling approaches [19], [20]. We use VITRUVIUS [21], which belongs in the latter category.

B. Co-Evolution of Models

Our work builds upon the co-evolution of models. Hebig, Khelladi, and Bendraou [22] give an overview of approaches for the co-evolution of metamodels and models. Cicchetti, Ruscio, Eramo, *et al.* [23] propose one of the earliest approaches to automate this co-evolution. Our approach is designed for the automated co-evolution of interdependent models and artefacts of industrial tools, thus not for the co-evolution of metamodels. Di Ruscio, Iovino, and Pierantonio [3] discuss the importance of co-evolving, among other things, interdependent tools.

Consistency preservation [1] and model repair [2] can be used for model co-evolution. Macedo, Jorge, and Cunha [24] give an overview of such approaches. Single-underlying models [16] combine multiple models into one single source of truth, thus enabling both multi-model consistency preservation and multi-view modeling. Meier, Werner, Klare, *et al.* [20] compare different SUM-based approaches, examples are VITRUVIUS [21] and *NAOMI* [25]. We use the former for our approach. We also relate to model synchronization, and concurrent editing [26], [27], as some of the challenges we identify are also present in these research areas.

C. MDE Adoption

As we aim at enabling the use of legacy software with model-driven consistency preservation to enable co-evolution between models and tools we consider approaches investigating MDE adoption. Bucaioni, Dimic, Gålnander, *et al.* [28] discuss how to transfer academic model-based methodology to automotive processes. Whittle, Hutchinson, Rouncefield, *et al.* [29], [30] discuss the industrial adoption of MDE and tool-related issues affecting it. They conclude that MDE tools should be designed to match the people and organizations and not vice versa. Moreover, they argue for focusing more on (improving) processes, less on (building) tools. Hence, our approach is designed as a process that does not require altering existing tools. Rather, it is designed as a non-intrusive way to enable co-evolution for industrial tools. In contrast to research enabling generic MDE-compatibility of legacy software, such as Ecoreification [4] or the code-first approach by Boronat [5], we focus on the artifacts of such software.

D. Model Transformations

Model transformations are a well-researched subject [31]–[33] and are often used for model co-evolution [34]. Our approach relies explicitly on incremental model transformations [13], [35], which updates a target model when a source model is changed. Kusel, Etzlstorfer, Kapsammer, *et al.* [36] provide a comprehensive overview of different incremental approaches. For the co-evolution of multiple models, transformation networks [37] or multiary transformations [38]–[40] can be employed. These approaches might be required for complex industrial tools that have interdependencies with multiple models.

E. Model Comparison

Model comparison [41], [42] matches model elements across models and calculates the differences between the models [43]. Our approach uses model comparison to derive the changes between two view states. One of the state-of-the-art tools is EMFCompare [44], [45]. It matches model elements leveraging similarity metrics and is customizable to specific needs. There are also metamodel-independent approaches such as DSMDiff [46]. Addazi, Cicchetti, Di Rocco, *et al.* [47] extended EMFCompare with semantic matching.

III. APPROACH

To enable the co-evolution of models and artefacts of industrial tools, we treat the artefacts of these tools as external views on a model in the system.

Definition 1: An *external tool* is a black-box software system that reads and persists its state in a specific format, as only means of exposing its state. Consider, for instance, a legacy software application that persists its state through XML files. Industrial software applications are such external tools.

Definition 2: An *external view* is a view that encapsulates an external tool. The external tool permits showing one or multiple target models upon which the external view is defined through the external view. An external view is predefined and usually a partial view of the system. It acts as an adapter between the models and the external tools, thus bridging the gap between them.

Definition 3: Like an ordinary viewtype, an *external viewtype* is the metamodel that specifies an external view. However, it additionally includes a transformation that allows generating views from the states of an external tool.

We divide the co-evolution problem into two parts: First, the model-view consistency between the external tool and the interdependent model. Second, the co-evolution of all models in the system. The latter can be realized using existing consistency preservation approaches, for instance, those based on

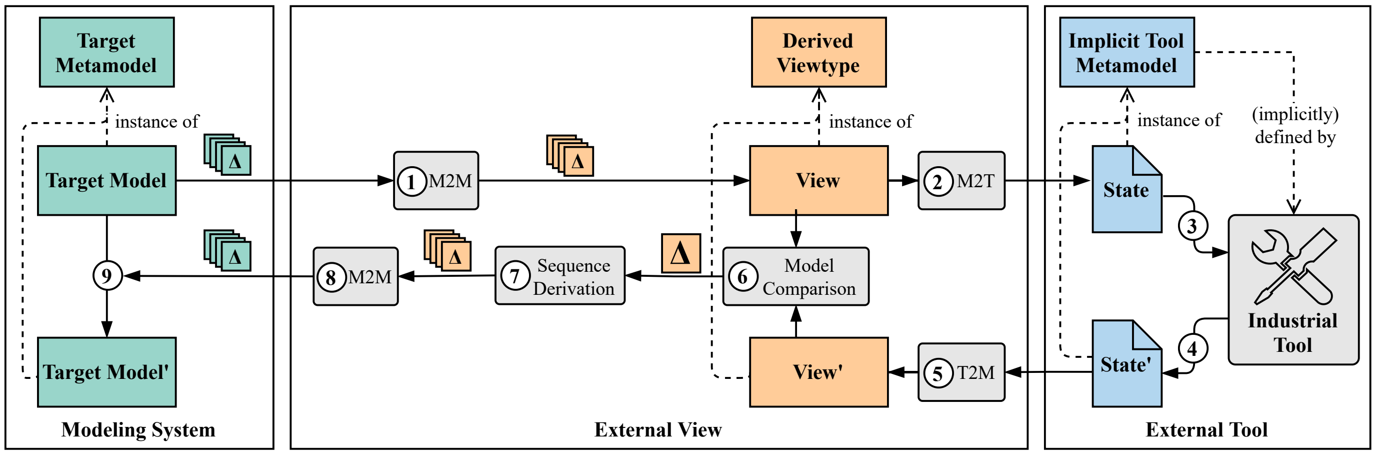

Fig. 1. Approach for the integration of external tools through external views to allow the co-evolution of models with the persisted states of the external tools.

model transformations, such as transformation networks [48] or multiary transformations [38]. To realize model-view consistency, we need explicit support for view definitions. There are multiple view-based approaches for consistency preservation that thus both allow multi-model co-evolution and view definition [20]. In the following, we propose our envisioned approach for the integration of external tools through external views.

A. Integrating External Tools

To enable model-view consistency, three steps are required: First, integrating the external tool through the creation of an external view. Second, initially synchronizing the external view with the interdependent model if initial artefacts or models exist. Third, keeping the external view consistent with the interdependent model whenever one is changed. In our approach, we distinguish three subsystems:

1) The *modeling system* contains all (formal and thus MDE-conforming) models. This can either be a single (mega-) model or multiple models.
2) The *external tool* contains the industrial tool and its artefact that are interdependent with (parts of) the modeling system. These artefacts conform to an *implicit* metamodel, which is inherently defined by the tool itself.
3) The *external view* describes a view on one or more models in the modeling system. The external view also contains a transformation for the persisted states of the external tool.

In a multi-model scenario, a specific tool might require an external view based on multiple models, as the artefacts of that tool might be interdependent with multiple models. For the sake of simplicity, we assume the single-model case in the remainder of this section, while the multi-model case is discussed in subsection IV-E. We also assume the external tool persists its state in a semi-structured file format such as XML. However, the approach can be adapted to support different ways of tool persistence.

B. Constructing External Views

In order to integrate the external tool into the modeling system, the external view acts as an adapter and ensures two properties: First, the view is provided in an MDE-conforming format, meaning instances of an explicit viewtype instead of a legacy format such as semi-structured files (without an explicit and formal metamodel). Second, external changes to the view are published to the modeling system as sequences of atomic changes instead of state-based differences.

To construct external views for a tool, we first need an external viewtype. This either be (semi-)automatically derived from the tool or built manually. It is an explicit metamodel for the tool. Then, we need a text-to-model transformation to instantiate views from our viewtype for persisted states of the external tool. Vice versa, we need a model-to-text transformation to generate states from views. Thus, we fulfill the first property. Last, we need incremental model-to-model transformations to transform between the external view and the target model to fulfills the second property. These transformations can either be manually constructed or derived (semi-) automatically.

Figure 1 illustrates our approach to integrate an external tool as an external view on the modeling system. It shows the three subsystems and how to keep them consistent. The external view forms a bridge between the modeling system and the external tool. The transformations (steps 1, 2, 5, and 8) connect to the external model and the external tool.

C. Model-View Consistency

Figure 1 shows both how changes can be propagated to the external tool and vice versa. When the target model is changed (either by a user or automatically by a consistency preservation mechanism), the external view is updated accordingly. Next, the tool state is regenerated from the external view. The user can then read the state with the external tool and make manual changes. When the user then saves these changes, the external view is regenerated from the tool. Then, the target model is updated from the external view. In detail, our approach realizes this as follows:

Target Model to External View: When the target model is modified, the changes are propagated to the external view with an incremental model-to-model transformation (see step 1 in Figure 1). If the target metamodel and the derived viewtype are identical, this is trivial. However, if they differ, the changes need to be transformed accordingly. This transformation needs to be incremental to avoid overriding changes in the external view.

External View to External Tool: When the view is updated, the persisted state of the external tool is regenerated utilizing a model-to-text transformation (step 2). This state can be read by the tool (step 3) to display its information within the tool. We do not require an incremental transformation for this step, as the external view is always kept up to date with the tool.

External Tool to External View: Changes made with the tool need to be persisted (step 4) to regenerate the view. This is done with a text-to-model transformation which uses the changed persisted state to regenerate the external view (step 5). As the external tool does not publish atomic changes, this transformation cannot be incremental.

External View to Target Model: To update the target model, two view states are used. The regenerated view (*View'*) as well as the previous view from before the regeneration (*View*). We then use model comparison (e.g. EMFCompare [45]) to calculate the state-based difference between these states (step 6). However, to support delta-based model-view consistency, we need a sequence of atomic changes that describe the difference between the two view states. As the real change sequence is not published by the external tool, we need to derive an estimated sequence (step 7). Last, we use an incremental model-to-model transformation (step 8) to update the target model from the changed view. The transformation being incremental enables updating the target model without overwriting concurrent changes in the modeling system.

Note that that we can use a single bidirectional transformation to both step (step 1) and (step 8), and another single bidirectional transformation for both step (step 3) and (step 4).

IV. Challenges

We identify the following fundamental challenges of the co-evolution of models with artefacts of industrial tools:

A. Viewtype Construction

To transform between the external view and the modeling system, an explicit viewtype is required. This viewtype needs to be derived from the external tool. While most industrial software applications do not use an explicit metamodel, they use some specification to persist their internal states. These implicit tool metamodels allow deriving the viewtype. In the best case, this is an XSD schema or a similar specification that allows deriving a viewtype automatically. However, in the worst case, the persistence rules are implicitly encoded in the code of the external tool and thus not accessible. In that case, the viewtype needs to be constructed based on indirect accounts, such as user documentation or tool experts. Essentially, the challenge is to find a generic process for deriving adequate viewtypes depending on the implicit tool metamodel. This could be done either manually or semi-automated.

B. Change Sequence Derivation

We regenerate the external view from the persisted state since the external tool does not expose changes in a delta-based manner. Consequently, the external view needs to provide fine-grained changes to be compatible with delta-based consistency preservation. It thus derives the state-based difference through model comparison (see step 6 in Figure 1). However, the external view then needs to derive a valid change sequence that reflects the hidden changes made with the external tools. While the change sequence does not need to be identical to the original change sequence, it needs to be *admissible*, meaning it has the same effect on the target models as the original changes would have [49]. Moreover, to understand the intent of the external changes, atomic changes should be grouped semantically, meaning atomic changes belonging to the same semantic change made in the external tool by a user should be grouped.

C. Tool Synchronization Frequency and Modality

For traditional software applications, there is a wide range in *how* and *how often* internal states are persisted and thus exposed. While some tools may save changes instantly, some may only save periodically, while others may only save when the user explicitly decides to do so. This limits how often changes in the external view are synchronized with the modeling system. Furthermore, not every persisted state might be valid or well-formed and thus meant to be synchronized. Thus, this leads to challenges: First, the external view needs to derive change sequences based on large state differences. Second, the external view needs to decide if a persisted state even should be synchronized, which can require user input.

D. Concurrent Editing

When both the external view and one or more models are concurrently modified, conflicting changes must be resolved. While concurrent modification is also a challenge in model synchronization itself [27], [50]–[52], it is amplified for external views due to the synchronization frequency. Between the infrequent synchronization of changes to the view, the modeling system can be modified. While the modeling system will synchronize internally, the state of the external tool cannot be regenerated whenever a user is using the tool since that would override changes made by the user. Later, when the tool persists the internal state (see step 4 in Figure 1), the external view will be regenerated (see step 5). All the changes in the modeling system since the last model-view synchronization are now potentially conflicting. This poses the challenge of dealing with conflicting changes. While some atomic changes can be synchronized, others might need to be rejected, requiring the user to solve the conflicts manually or redo the changes altogether. Especially when the modeling system contains multiple models, internal changes might need to be prioritized over the external changes made in the tool to avoid interfering with the consistency preservation in the modeling system.

E. Multi-Model Interdependency

As discussed previously, the modeling system can contain multiple interdependent models. The artefacts of an external tool can be interdependent with multiple models as well. Especially when a single model in the system cannot entirely represent information persisted by an external tool, the external view needs to be based on multiple models. Thus, we require multi-model-view consistency. This can be achieved with multiple incremental transformations but may lead to complex compatibility issues as observed in transformation networks [53]. In a multi-model scenario, a specific tool might require an external view that is based on multiple models, as the artefacts of that tool might be interdependent with multiple models. For the sake of simplicity, we assume the single-model case in the remainder of this section, meaning the external view is based on a single target model as the external is interdependent with exactly one model in the modeling system. The multi-model case is discussed later.

F. Lack of Unique Identifiers

Unique identifiers are used in models to distinguish model elements during their lifetime. Thus, they are essential for consistency preservation approaches. However, many external tools might not use unique identifiers in their persisted states. Thus, an external view needs to ensure compatibility with states without unique identifiers. Since similarity-based model comparison strategies perform significantly worse than identifies based strategies [49], domain-specialized post-processing needs to be applied during the change sequence derivation (see step 7 in Figure 1). Another solution could be enriching the views with unique identifiers after regenerating them from the states (see step 5). However, this needs to be deterministic across all possible persisted states.

G. Private Data

Private data is a subset of the information persisted by the external tool that cannot be represented in the modeling system, as it is not interdependent with the modeling system. Tool-specific layout information for the graphical representation is such private data. It is essential for the external tool but cannot be transferred in the modeling system. However, this information must be preserved, which means it cannot be overwritten when updating the external view. Moreover, this information might need to be explicitly generated for newly created model elements if the tool expects such information for all model elements.

V. Planned Evaluation

We plan on evaluating our approach with the Ecore-based Vitruvius framework [21]. Vitruvius combines models into a virtual single-underlying model (V-SUM) employing transformation networks. In the V-SUM, consistency is preserved in a delta-based way, meaning incremental model transformations transform fine-grained sequences of atomic changes to the source model to the resulting change sequence for the target model. These transformations are written in the Reactions language [54], an imperative, domain-specific language for consistency preservation. Vitruvius enables view-based development, as views can be defined upon the V-SUM. Thus, the V-SUM is the modeling system from our approach, and external views are integrated by connecting them to one or more target models in the V-SUM with Vitruvius we want to evaluate the approach in two domains:

Software Engineering Models and Tools: UML models, architecture models, and code share interdependent information. With Vitruvius we can keep these models persistent. However, many of the common tools and editors are not compatible with the respective metamodels. Thus, we want to conduct a case study based on models and external tools used in software development. We plan on integrating an external tool into our modeling systems leveraging external views. Potential tools are, for example, proprietary UML editor tools for software architecture design.

Automotive Models and Tools: During the development of automotive systems, a plethora of heterogeneous models is used. This includes, but is not limited to, CAD models, AUTOSAR architecture models, hardware network topology models, hardware component architectures, models for electric circuits, and models for the wiring topology. These models are developed in different ways with different tools. These tools are often long-living and have limited compatibility with other tools. These automotive systems are large-scale, which is where model-driven approaches excel. We thus plan a comprehensive case study based on multiple models and multiple industrial legacy tools from the automotive domain to evaluate our co-evolution approach based on external views.

VI. Conclusion

As modern software systems development comprises multiple interdependent models that change throughout development, there is a need for managing the co-evolution of these interdependent models. Although this is challenging in its own right, the continued use of legacy or industrial tools for software development limits current approaches for model consistency preservation. To cope with the integration of legacy industrial tools, we propose external views to encapsulate external tools and their state-based persistence in files. In particular, we propose an approach for the co-evolution of artefacts of tools with models leveraging consistency preservation and a general architecture and a development process for external views. Moreover, we identified, among others, the change sequence derivation, the synchronization frequency and modality, concurrent editing, and private data as key challenges for the co-evolution of artefacts of industrial tools with multiple models. Besides that, we outlined our approach's planned implementation and evaluation based on the existing model consistency preservation framework–Vitruvius and legacy tools from the software engineering and the automotive domain. With this research, we not only approach the integration of external tools into model co-evolution but also aim to improve the adoption of MDE beyond the software engineering domain.

References

[1] W. Torres, M. G. J. van den Brand, and A. Serebrenik, "A systematic literature review of cross-domain model consistency checking by model management tools," *Softw. Syst. Model.*, vol. 20, no. 3, pp. 897–916, 2021. DOI: 10.1007/s10270-020-00834-1.

[2] P. Stünkel, H. König, A. Rutle, and Y. Lamo, "Multi-model evolution through model repair," *Journal of Object Technology*, vol. 20, no. 1, 1:1–25, 2021, Workshop on Models and Evolution (ME 2020). DOI: 10.5381/jot.2021.20.1.a2.

[3] D. Di Ruscio, L. Iovino, and A. Pierantonio, "What is needed for managing co-evolution in mde?" In *Proceedings of the 2nd International Workshop on Model Comparison in Practice*, 2011, pp. 30–38. DOI: 10.1145/2000410.2000416.

[4] H. Klare, E. Burger, M. E. Kramer, M. Langhammer, T. Sağlam, and R. Reussner, "Ecoreification: Making arbitrary java code accessible to metamodel-based tools," in *Proceedings of the ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017)*, 2017. DOI: 10.1109/MODELS.2017.30.

[5] A. Boronat, "Code-first model-driven engineering: On the agile adoption of mde tooling," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 874–886. DOI: 10.1109/ASE.2019.00086.

[6] C. Atkinson, D. Stoll, and P. Bostan, "Orthographic Software Modeling: A Practical Approach to View-Based Development," in *Evaluation of Novel Approaches to Software Engineering*, vol. 69, 2010, pp. 206–219. DOI: 10.1007/978-3-642-14819-4_15.

[7] T. Goldschmidt, S. Becker, and E. Burger, "Towards a tool-oriented taxonomy of view-based modelling," in *Proceedings of the Modellierung 2012*, vol. P-201, 2012, pp. 59–74.

[8] E. Burger, "Flexible Views for View-Based Model-Driven Development," in *Proceedings of the 18th international doctoral symposium on Components and architecture*, 2013, pp. 25–30. DOI: 10.1145/2465498.2465501.

[9] H. Brunelière, J. Garcia Perez, M. Wimmer, and J. Cabot, "EMF Views: A View Mechanism for Integrating Heterogeneous Models," in *34th International Conference on Conceptual Modeling (ER 2015)*, 2015.

[10] F. Bancilhon and N. Spyratos, "Update semantics of relational views," *ACM Trans. Database Syst.*, vol. 6, no. 4, pp. 557–575, 1981. DOI: 10.1145/319628.319634.

[11] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt, "Combinators for bi-directional tree transformations: A linguistic approach to the view update problem," *SIGPLAN Not.*, vol. 40, no. 1, pp. 233–246, 2005. DOI: 10.1145/1047659.1040325.

[12] ——, "Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem," *ACM Trans. Program. Lang. Syst.*, vol. 29, no. 3, 17–es, 2007. DOI: 10.1145/1232420.1232424.

[13] Z. Diskin, Y. Xiong, K. Czarnecki, H. Ehrig, F. Hermann, and F. Orejas, "From state- to delta-based bidirectional model transformations: The symmetric case," in *Model Driven Engineering Languages and Systems*, vol. 6981, 2011, pp. 304–318. DOI: 10.1007/978-3-642-24485-8_22.

[14] H. Giese and R. Wagner, "Incremental model synchronization with triple graph grammars," in *Model Driven Engineering Languages and Systems*, 2006, pp. 543–557.

[15] ——, "From model transformation to incremental bidirectional model synchronization," *Software & Systems Modeling*, vol. 8, no. 1, pp. 21–43, 2009. DOI: 10.1007/s10270-008-0089-9.

[16] C. Atkinson and D. Stoll, "Orthographic modeling environment," in *Fundamental Approaches to Software Engineering*, 2008, pp. 93–96. DOI: 10.1145/2489861.2489862.

[17] C. Tunjic and C. Atkinson, "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*, 2015, pp. 55–58. DOI: 10.1145/2802059.2802066.

[18] H. Brunelière, E. Burger, J. Cabot, and M. Wimmer, "A Feature-based Survey of Model View Approaches," *Software & Systems Modeling*, 2017. DOI: 10.1007/s10270-017-0622-9.

[19] A. Cicchetti, F. Ciccozzi, and A. Pierantonio, "Multi-view approaches for software and system modelling: A systematic literature review," *Software & Systems Modeling*, vol. 18, pp. 3207–3233, 2019. DOI: 10.1007/s10270-018-00713-w.

[20] J. Meier *et al.*, "Classifying approaches for constructing single underlying models," in *Model-Driven Engineering and Software Development*, 2020, pp. 350–375. DOI: 10.1007/978-3-030-37873-8_15.

[21] H. Klare, M. E. Kramer, M. Langhammer, D. Werle, E. Burger, and R. Reussner, "Enabling consistency in view-based system development — the vitruvius approach," *Journal of Systems and Software*, vol. 171, p. 110 815, 2021. DOI: 10.1016/j.jss.2020.110815.

[22] R. Hebig, D. E. Khelladi, and R. Bendraou, "Approaches to co-evolution of metamodels and models: A survey," *IEEE Transactions on Software Engineering*, vol. 43, no. 5, pp. 396–414, 2017. DOI: 10.1109/TSE.2016.2610424.

[23] A. Cicchetti, D. D. Ruscio, R. Eramo, and A. Pierantonio, "Automating co-evolution in model-driven engineering," in *2008 12th International IEEE Enterprise Distributed Object Computing Conference*, 2008, pp. 222–231. DOI: 10.1109/EDOC.2008.44.

[24] N. Macedo, T. Jorge, and A. Cunha, "A Feature-based Classification of Model Repair Approaches," *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 615–640, 2017. DOI: 10.1109/TSE.2016.2620145.

[25] T. Denton, E. Jones, S. Srinivasan, K. Owens, and R. W. Buskens, "Naomi – an experimental platform for multi-modeling," in *Model Driven Engineering Languages and Systems*, 2008, pp. 143–157.

[26] F. Orejas, E. Pino, and M. Navarro, "Incremental concurrent model synchronization using triple graph grammars," in *Fundamental Approaches to Software Engineering*, 2020, pp. 273–293. DOI: 10.1007/978-3-030-45234-6_14.

[27] Y. Xiong, H. Song, Z. Hu, and M. Takeichi, "Synchronizing concurrent model updates based on bidirectional transformation," *Software and Systems Modeling*, vol. 12, no. 1, pp. 89–104, 2013. DOI: 10.1007/s10270-010-0187-3.

[28] A. Bucaioni, V. Dimic, M. Gålnander, H. Lönn, and J. Lundbäck, "Transferring a model-based development methodology to the automotive industry," in *2021 22nd IEEE International Conference on Industrial Technology (ICIT)*, vol. 1, 2021, pp. 762–767. DOI: 10.1109/ICIT46573.2021.9453680.

[29] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial adoption of model-driven engineering: Are the tools really the problem?" In *Model-Driven Engineering Languages and Systems*, 2013, pp. 1–17. DOI: 10.1007/978-3-642-41533-3_1.

[30] ——, "A taxonomy of tool-related issues affecting the adoption of model-driven engineering," *Softw. Syst. Model.*, vol. 16, no. 2, pp. 313–331, 2017. DOI: 10.1007/s10270-015-0487-8.

[31] P. Stevens, "A landscape of bidirectional model transformations," in *Generative and Transformational Techniques in Software Engineering II: International Summer School, 2007, Braga, Portugal. Revised Papers*. 2008, pp. 408–424. DOI: 10.1007/978-3-540-88643-3_10.

[32] L. Samimi-Dehkordi, B. Zamani, and S. Kolahdouz-Rahimi, "Bidirectional model transformation approaches a comparative study," in *2016 6th International Conference on Computer and*

Knowledge Engineering (ICCKE), 2016, pp. 314–320. DOI: 10.1109/ICCKE.2016.7802159.

[33] S. Hidaka, M. Tisi, J. Cabot, and Z. Hu, "Feature-based classification of bidirectional transformation approaches," *Software & Systems Modeling*, vol. 15, no. 3, pp. 907–928, 2016. DOI: 10.1007/s10270-014-0450-0.

[34] N. Kahani, M. Bagherzadeh, J. R. Cordy, J. Dingel, and D. Varró, "Survey and classification of model transformation tools," *Software & Systems Modeling*, vol. 18, no. 4, pp. 2361–2397, 2018. DOI: 10.1007/s10270-018-0665-6.

[35] H. Giese and R. Wagner, "From model transformation to incremental bidirectional model synchronization," *Software & Systems Modeling*, vol. 8, no. 1, pp. 21–43, 2008. DOI: 10.1007/s10270-008-0089-9.

[36] A. Kusel *et al.*, "A survey on incremental model transformation approaches," in *ME 2013 – Models and Evolution Workshop Proceedings*, 2013, pp. 4–13.

[37] P. Stevens, "Maintaining consistency in networks of models: Bidirectional transformations in the large," *Software and Systems Modeling*, vol. 19, no. 1, pp. 39–65, 2020. DOI: 10.1007/s10270-019-00736-x.

[38] A. Cleve, E. Kindler, P. Stevens, and V. Zaytsev, "Multidirectional Transformations and Synchronisations (Dagstuhl Seminar 18491)," *Dagstuhl Reports*, vol. 8, no. 12, pp. 1–48, 2019. DOI: 10.4230/DagRep.8.12.1.

[39] P. Stünkel, H. König, Y. Lamo, and A. Rutle, "Multimodel Correspondence Through Inter-model Constraints," in *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming*, 2018, pp. 9–17. DOI: 10.1145/3191697.3191715.

[40] Z. Diskin, H. König, and M. Lawford, "Multiple Model Synchronization with Multiary Delta Lenses," in *Fundamental Approaches to Software Engineering*, 2018, pp. 21–37. DOI: 10.1007/978-3-319-89363-1_2.

[41] D. S. Kolovos, D. D. Ruscio, A. Pierantonio, and R. F. Paige, "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, pp. 1–6. DOI: 10.1109/CVSM.2009.5071714.

[42] M. Stephan and J. 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, 2013, pp. 265–277. DOI: 10.5220/0004311102650277.

[43] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, *An Introduction to Model Versioning*, M. Bernardo, V. Cortellessa, and A. Pierantonio, Eds. Springer Berlin Heidelberg, 2012, pp. 336–398. DOI: 10.1007/978-3-642-30982-3_10.

[44] C. Brun and A. Pierantonio, "Model differences in the eclipse modeling framework," *UPGRADE, The European Journal for the Informatics Professional*, vol. 9, no. 2, pp. 29–34, 2008.

[45] E. Foundation, *EMF Compare*. [Online]. Available: https://www.eclipse.org/emf/compare (visited on 07/22/2021).

[46] Y. Lin, J. Gray, and F. Jouault, "Dsmdiff: A differentiation tool for domain-specific models," *European Journal of Information Systems - EUR J INFOR SYST*, vol. 16, pp. 349–361, 2007. DOI: 10.1057/palgrave.ejis.3000685.

[47] L. Addazi, A. Cicchetti, J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, "Semantic-based model matching with emfcompare," in *10th Workshop on Models and Evolution*, 2016, pp. 40–49. DOI: 10.1145/3417990.3421999.

[48] P. Stevens, "Bidirectional Transformations in the Large," in *ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2017, pp. 1–11. DOI: 10.1109/MODELS.2017.8.

[49] J. W. Wittler, "Derivation of Change Sequences from State-Based File Differences for Delta-Based Model Consistency," to appear, Master's Thesis, Karlsruhe Institute of Technology (KIT), 2021, 73 pp.

[50] F. Hermann, H. Ehrig, C. Ermel, and F. Orejas, "Concurrent model synchronization with conflict resolution based on triple graph grammars," in *Fundamental Approaches to Software Engineering*, 2012, pp. 178–193. DOI: 10.1007/978-3-642-28872-2_13.

[51] N. Weidmann and G. Engels, "Concurrent model synchronisation with multiple objectives," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2021, pp. 1097–1105. DOI: 10.1145/3449639.3459283.

[52] S. Ebert, T. Kluge, and S. Götz, "Resolving synchronization conflicts in role-based multimodel-synchronization environments," in *Proceedings of the 13th ACM International Workshop on Context-Oriented Programming and Advanced Modularity*, 2021, pp. 1–8. DOI: 10.1145/3464970.3468412.

[53] T. Sağlam and H. Klare, "Classifying and avoiding compatibility issues in networks of bidirectional transformations," in *STAF 2021 Workshop Proceedings: 9th International Workshop on Bidirectional Transformations*, accepted, to appear, 2021.

[54] H. Klare, "Designing a Change-Driven Language for Model Consistency Repair Routines," Master's Thesis, Karlsruhe Institute of Technology (KIT), 2016. DOI: 10.5445/IR/1000080138.