Multi-model Consistency Preservation

Heiko Klare
Institute for Program Structures and Data Organization (IPD)
Karlsruhe Institute of Technology, Germany
heiko.klare@kit.edu

ABSTRACT

Modern software systems are developed using multiple models to represent different properties of the system. Since these models contain dependent information, keeping them consistent is crucial for producing correctly operating software. This process can be automated with model transformations, and has been researched mainly for *binary transformations* that define consistency preservation for pairs of metamodels. If more than two metamodels are involved, the combination of several binary transformations leads to problems such as *interoperability issues* and occurring *trade-off decisions*. Therefore, we investigate these problems in *multi-model consistency preservation* in our thesis. We first analyze the interoperability of binary transformations that have been independently developed and derive patterns for *non-intrusive transformation interoperability*. Furthermore, we introduce an approach for decomposing consistency relations and *concept metamodels* that make relations explicit. This is supposed to reduce the necessity for trade-off decisions regarding challenges such as *compatibility* of dependent transformations and *modularity* of used metamodels and transformations, as well as *comprehensibility* and *evolvability*. The proposed benefit of our approach is that independently developed binary transformations can be combined, omitting interoperability issues without the necessity to know about each other, to achieve multi-model consistency. We will evaluate our approach with case studies from component-based and object-oriented software, as well as with industrial case studies for embedded automotive software and production systems.

CCS CONCEPTS

• **Software and its engineering → Model-driven software engineering**; **Consistency**; **Abstraction, modeling and modularity**; *Software system models*; *Software evolution*; Correctness;

KEYWORDS

consistency management, consistency preservation, multi-model consistency, model transformation, domain-specific languages

ACM Reference Format:
Heiko Klare. 2018. Multi-model Consistency Preservation. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18 Companion), October 14–19, 2018, Copenhagen, Denmark*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3270112.3275335

MODELS '18 Companion, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-5965-8/18/10…$15.00
https://doi.org/10.1145/3270112.3275335

1 INTRODUCTION

Model-driven software development proposes the usage of models as primary artifacts of the development process [38]. Those models describe different system properties for the interests of specific stakeholders, known as *multi-view modelling*, or at different abstraction levels, representing refinements. In both cases, the models describe the same system and are thus not disjoint but contain redundant or dependent information. Developers must be aware of those dependencies to ensure that models are modified consistently.

In large software systems, a single developer cannot know about all dependencies [35], in the following referred to as *consistency relations*, which inevitably leads to inconsistencies. Therefore, automated mechanisms that preserve consistency according to those consistency relations are necessary. For that purpose, incremental, bidirectional model transformations are commonly used. However, most research considers *binary transformations*, restricted to pairs of models, and does not explicitly consider consistency between more than two models [39], which we refer to as *multi-model consistency*. Model transformations can either be specified imperatively or declaratively. They differ in who operalizationalizes the preservation of constraints that have to hold, in the first case the transformation developer and in the second case an automated mechanism of the transformation language. This is why we do not explicitly distinguish these approaches, as all problems apply to both approaches and only different roles have to deal with them.

Although it is possible to combine binary transformations by transitively executing them, it is yet unclear what problems may arise from that, especially if each transformation is developed independently and treated as a black box. We will exemplify this on the simple example in Figure 1, in which consistency relations define a mapping of a component in an *architecture description language* (ADL) to a class in object-oriented design, which is again represented by an implemented class in Java code. The name of the class is defined to be the component name with an "Impl" suffix (cf. [21]). When all these relations are expressed in transformations, it is, for example, possible that both transformations from ADL to Java, once over UML (R_1 and R_2) and once directly (R_3), create a Java class after creating an ADL component. We refer to that as an *interoperability problem*. The transformation specification would have to avoid an overwrite and therefore have to consider dependencies between transformations, using, for example, a shared trace model. In general, an interoperability problem is an unexpected behavior of transformations, which only occurs if they are executed transitively, but not if each is executed on its own.

Additionally, it is easy to see that combining binary transformations leads to trade-off decisions. The ternary relation can either be expressed by three binary transformations between all pairs of metamodels or by two binary transformations with the third being the combination of the two others. The first option leads to

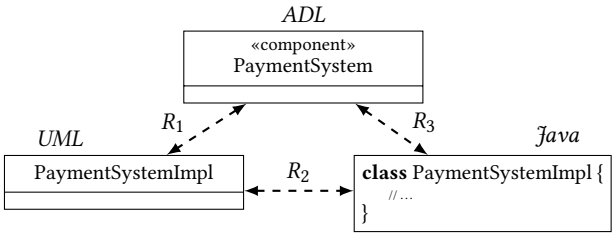


Figure 1: Example models with binary consistency relations

redundancies in the specifications, as each pair of transformations has to have an equal semantics than the third. For example, the ADL to Java transformation for R_3 must be equal to the combination of the transformations ADL to UML (R_1) and UML to Java (R_2). Consequently, those transformations may be incompatible if not correctly defined, e.g., by leaving out the suffix addition in the transformation for R_3. An alternative is to omit the transformation for R_3 by transitively executing the two others. However, in this case, modularity is reduced, because it is not possible to use only Java and the ADL to develop a specific system and omit the UML. We refer to this as *specification trade-offs*.

Instead of developing approaches to express multiary consistency relations, there are reasons to adhere to binary transformations, and to research their combinability. As stated by Stevens [39], it is hard enough to think about binary relations. Additionally, each domain expert, who specifies transformations, will usually only have knowledge about the relations between two or at most a rather limited set of metamodels. We therefore plan to make the following contributions to research on multi-model consistency preservation:

Transformation interoperability. When several binary transformations are developed independently, they must be combinable in a black box manner, introduced as the *interoperability problem*. We will therefore identify problems that can arise from that combination and develop a catalog of patters that can be followed by the transformation developer or language to achieve *non-intrusive* interoperability of binary transformations.

Decomposition of consistency relations. The usage of binary transformations for multi-model consistency preservation leads to *specification trade-offs* regarding essential challenges. We will provide a classification of those challenges and investigate the influence of the way in which transformations are specified on them. We will especially investigate how consistency relations can be decomposed into independent subsets, as this allows a partial optimization regarding those challenges.

Make common concepts explicit. Metamodels often represent the same concepts in different ways. As another contribution to reduce *specification trade-offs*, we propose an approach to make these common concepts explicit to improve comprehensibility of transformations and to improve their modular reuse.

Throughout this paper, we use a simplified notation for metamodels and heir consistency relations to ease their illustration. We consider metamodels to be sets of elements and consistency relations to be sets of symmetric, binary relations between those elements. To ease the representation of combinations of consistency relations, we define the concatenation operation for two consistency relations R_1 and R_2 as:

$$R_1 \oplus R_2 := \{(x, y) \mid \exists t : x\, R_1\, t\, R_2\, y\}.$$

This is the subset of the transitive closure of two relations that contains only the relations transitively defined over R_1 and R_2. It can be also expressed as the natural join of R_1 and R_2 with an additional projection that removes the common elements of both relations. The operator is commutative since the relations are assumed symmetric.

2 RELATED WORK

Model consistency preservation, also referred to as *model repair*, is an active field of current research. Nevertheless, most approaches are restricted to consistency between pairs of models [39]. The kinds of dependencies between multiple models and types of inconsistencies were discussed by Kolovos et al. [16]. A summary and classification of model consistency approaches, also regarding their multi-model support, was presented by Macedo et al. [28].

As stated by Stevens [39], it is reasonable to target multi-model consistency by combining binary transformations. Especially incremental model transformation languages can be applied by executing the transformations transitively. They were surveyed by Kusel et al. [20], including the Atlas Transformation Language (ATL) [14, 46], VIATRA [4] and Triple Graph Grammars (TGGs) [3, 2]. For TGGs, initial concepts for supporting multiple models were proposed [41, 40]. Another transformation-based tool is *Echo* [27]. It is based on *Alloy* [25], which uses QVT-R [32] and model finding to repair inconsistencies. For QVT-R, challenges and steps towards supporting transformations between multiple model were discussed [26]. Kramer proposed an approach combining a language for declarative mappings between metamodels with a fallback language for imperative consistency repair [19, 15], developed in the context of the Vitruvius framework [18]. Except for the explicitly mentioned works, those approaches do not explicitly deal with challenges introduced by combining binary transformations. Another topic regarding transformations is *uncertainty*. Not all decision can be made automatically, e.g., whether a created Java class shall represent an ADL component or not. To handle this, different solutions can be calculated [9] or the developer can be asked for his intent [22].

Approaches regarding the combination of binary transformations are rather focused on composing transformations between the same two metamodels [42, 44, 43]. Additionally, those approaches do usually not consider transformations as black boxes, but provide intrusive composition operators that require adaptions of the composed transformations [37]. Some approaches also deal with processes for composition and simply assume interoperability [34].

An approach specific for consistency between different ADLs is DUALLy [29, 7]. It requires the specification of relations between concrete ADLs to a central, predefined ADL metamodel. DUALLy is based on a generic model consistency approach, which uses *answer set programming* (ASP) [6, 8] based on logical programming techniques. Such an approach of adding additional metamodels to represent consistency relations is also shortly discussed in [39].

There has also been some research on design patterns for transformations [12, 23]. They have been surveyed by Lano et al. [24], but mainly unify how specific kinds of consistency relations can be expressed in transformation languages and not on achieving non-intrusive transformation interoperability.

3 MULTI-MODEL CONSISTENCY

In this section, we discuss problems and proposed solutions regarding the combinability of binary transformations, starting with interoperability of independently developed transformations. Afterwards, we identify higher-level challenges arising from the combination of transformations, such as compatibility and modular reuse, and discuss their interdependencies and solvability. Finally, we present our proposed solutions to solve those challenges.

3.1 Binary Transformation Interoperability

Multi-model consistency preservation can be a achieved by combining binary transformations to graphs, with the transformations being executed transitively. Since all binary transformations are developed independently of each other, it is necessary that they interoperate properly in a *non-intrusive* way, thus without the necessity for the developer to understand and modify them, which we refer to as *black-box combination*.

Even under the assumption that, in contrast to our introductory motivation, all specifications are free of contradictions, it is easy to see that problems arise when combining binary transformations by transitively executing them. For example, consider the relations in Figure 1. If a component is added to the ADL, causing a UML class creation due to R_1, which in turn causes a Java class creation due to R_2, the transformation for relation R_3 does not know that an appropriate class was already created, if the transformations are treated as black boxes. Consequently, the transformation will create the same class again, which may override the existing one, depending on the implementation and execution order. A simple solution for this example would be to have all transformations use a common trace model and check for existing elements before creating them in a transformation. Nevertheless, independently developed transformations will usually not assume that other transformations may already have created corresponding elements. Additionally, the trace model must allow the transformation engine to retrieve transitive traces. However, it is unclear if transitive resolution of traces can always be performed, as it can depend on whether the transitive trace belongs the considered consistency relation or another.

As can be seen in the example, especially the correct handling of trace information in interdependent transformations has to be researched. This applies not only to element creations, but also other change types, such as attribute or reference changes, especially if they are multi-valued. In our thesis, we will therefore apply transitively executed binary transformations in different case studies to identify these and potential further problems. We then want to come up with a catalog of such problems together with solution patterns for them. For example, to avoid duplicate element creations, a simple pattern could be to always check for already existing traces for that consistency relation in the transformations. In consequence, the integration of those patterns into a transformation language or the application of them as a transformation developer is supposed to achieve black-box combinability of the transformations.

3.2 Challenges and Topology Trade-Offs

Even if independently developed transformations are interoperable, as discussed before, higher-level challenges remain when considering the way in which transformations express relations or occur if compatibility between the transformations is not given. We present already identified challenges in the following.

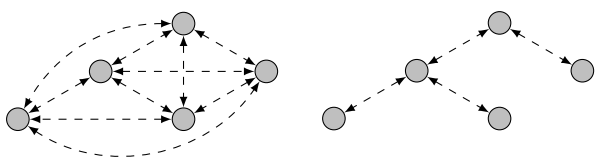
Compatibility Transformations preserve consistency, but also have to be consistent among themselves, i.e., they have to adhere to the same consistency relations, referred to as *compatibility*. If, in our example in Figure 1, the relation R_3 between ADL and Java is realized in addition to the transitive relation $R_1 \oplus R_2$ over UML, both relations must contain the same name attribute mapping. If the ADL to Java transformation adds an "Impl" suffix [21], whereas the transformations over UML omit that, they are *incompatible*. This can lead to propagation cycles due to alternating values, and to results depending on the transformation execution order. While *compatibility* concerns problems due to the realization of contradictory consistency relations, *interoperability* concerns potentially unexpected behavior although all transformations follow to the same consistency relations.

Modularity The development of different systems requires the usage of different metamodels to describe them. Therefore, transformations should be modular, so that an arbitrary selection of metamodels and transformations between them can be used within a concrete project. If, in our example, transformations are specified transitively across UML, it is impossible to use only the ADL and Java in a concrete project, omitting the UML.

Comprehensibility Maintainability of artifacts, including transformations, depends on their comprehensibility. Comprehensibility can be seen as the number of transformations to consider if a specific consistency relation shall be understood. Realizing consistency relations in transitive transformations can, for example, reduce comprehensibility, as a developer has to consider a set of transformations to understand a single consistency relation.

Evolvability Whenever new transformations shall be defined, e.g. because a new metamodel shall be used and transformations for it have to be defined, the effort should be as low as possible. This concerns the number of transformations to define and how many metamodels are involved in the transformations for one consistency relation if it is expressed transitively.

We will focus on compatibility and modularity, as they are crucial for the applicability of transformations. Defining binary transformations for a set of metamodels leads to a trade-off solution regarding those challenges, as they cannot be solved simultaneously. The intuitive way to define transformations is to express each relation between two metamodels in one transformation, which leads to a dense graph of transformations. In the extreme case, if all pairs of metamodels have non-empty consistency relations, the graph is complete, as shown in Figure 2a. In that case, modularity is high because each metamodel can be excluded without any drawback,



(a) Dense graph **(b) Tree**

Figure 2: Extreme examples for transformation topologies

Challenge	Dense Graph	Tree
Compatibility	-	++
Modularity	++	-
Comprehensibility	+	-
Evolvability	-	+

Table 1: Challenge fulfillment by transformation topology

but relations are likely to be incompatible, as, in the worst case, each relation is specified over $(n-1)!$ transformation paths if n metamodels are involved. While comprehensibility is high, as each relation is explicitly expressed, adding a metamodel requires to define up to $n-1$ transformations, implying high evolution effort.

Another extreme case is to have each consistency relation only defined over a single path in the transformations graph, which results in a tree of transformations, as shown in Figure 2b. In that case, compatibility of transformations is inherently given, as each relation is only defined once, but modularity is reduced, as only metamodels being leaves can be omitted. Comprehensibility is low, as each relation may be defined in a path of up to $n-1$ transformations, but evolvability is rather good, as each relation must only be defined once. We summarize that impact of the topology in Table 1.

A tree topology has the drawback that it is not always applicable. It requires that the transformation developers find a subset of all consistency relations between the metamodels that induces a tree and whose transitive closure contains all other relations. In general, such a subset cannot be found. Of three metamodels, there must always be one containing the overlapping information of the two others, as only then the transitive closure of two consistency relations subsumes the third. In the example in Figure 3, it is necessary that two relations are contained in the transitive closure of the others to get a tree that covers all relations.

In practice, the used topology will potentially be a mixture between those extremes. The natural way to foster the independent development of transformations would be to define one transformation for each consistency relation, resulting in a dense graph with a high potential for incompatible transformations. To deal with that, mechanisms that analyze compatibility between transformations could be researched. Nevertheless, high expressiveness of transformation languages allows only conservative approximations. In our thesis, we will therefore investigate approaches that result in tree-like specifications that directly imply compatibility between the transformations, but with increased *applicability* and *modularity*.

3.3 Decomposition of Consistency Relations

Defining a tree of transformations to preserve consistency between multiple models is difficult, because of three metamodels there must always be one containing all overlapping information of the two others. It is easy to see that this is hardly applicable in practice. Nevertheless, it may be sufficient to find subsets of independent consistency relations, of which each induces such a tree, instead of considering the whole set of relations between two metamodels. For example, consider the relations in Figure 4. R_3 represents a subset of the transitive relation $R_1 \oplus R_2$. In consequence, R_3 can be omitted, but the remaining consistency relations still form a graph. However, since R_4 is independent from the other relations, it can be treated separately from the others, resulting in two trees of independent consistency relations.

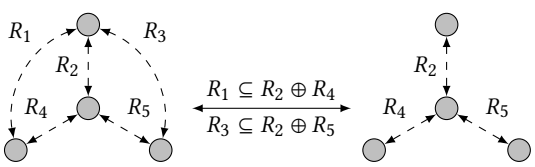


Figure 3: Reducing a consistency relation graph to a tree

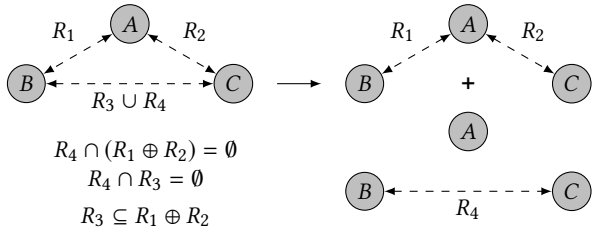


Figure 4: Exemplary decomposition of consistency relations

In our thesis, we will therefore analyze different case studies for the existence of such independent consistency relations, which allows to decompose them to independent trees. We will develop an appropriate representation of consistency relations, which is more sophisticated than the set notation used in this paper, and an analysis for independence of those relations. We will then investigate how that can be integrated into existing, declarative transformation languages. In consequence of that, we improve the *applicability* of a tree-based specification of transformations.

3.4 Making Common Concepts Explicit

Based on the decomposition of consistency relations, we propose another approach to achieve a tree structure of transformations, which inherently optimizes *modularity*. The idea bases on the insight that we can distinguish two kinds of consistency relations:

Descriptive consistency relations are "naturally" given when two metamodels represent common *concepts* redundantly or at least with dependent properties. This is, for example, the case for UML class models and Java realizing object orientation (OO).

Normative consistency relations prescribe consistency and do not exist "naturally". This is especially the case if metamodels represent different abstractions or domains of a system, which have no implicit relation, such as an ADL and Java.

While descriptive consistency relations between two metamodels are usually definite, such as those between OO design in UML and Java, normative consistency relations may vary depending on the project context. For example, several possible relations can be defined between an ADL and OO design, for example the realization of each component as a class or as a complete project [21].

Transformations for descriptive consistency relations implicitly encode the common concepts. Instead, we propose to make these common concepts explicit in so-called *concept metamodels (CMMs)* and define relations between them and the concrete metamodels. We illustrate this in Figure 5. The descriptive consistency relation

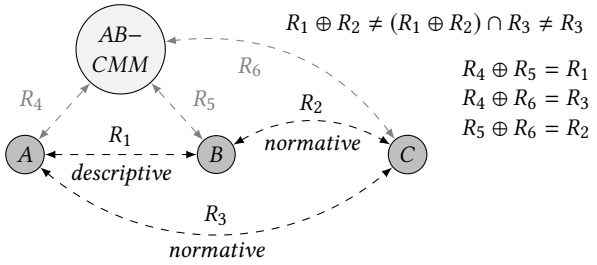


Figure 5: Definition of a concept metamodel

R_1 is converted into a CMM for the metamodels A and B with new relations R_4 and R_5 between the concrete metamodels and the CMM. The existing normative consistency relations R_2 and R_3 to metamodel C are replaced by a new relation R_6 to the CMM. The CMM and its consistency relations have to be appropriately defined to replace the original ones, as depicted in Figure 5. It will be part of our research to figure out how to define such a CMM, so that it can also be combined with other metamodels. While it basically has to contain the common concepts of the metamodels sharing a descriptive consistency relations, it may also need to contain additional information depending on consistency relations to other metamodels, which are not known a-priori.

This approach addresses all identified challenges and solves them under the assumption that all consistency relations are described using CMMs. We achieve inherent transformation compatibility by avoiding more than one transformation path between two metamodels by design. Since metamodels are only coupled across CMMs and thus represent leaves of the transformation tree, any subset of them can be selected, maximizing modularity. Finally, we assume that making common concepts explicit improves comprehensibility.

The most crucial part of this approach is the necessity to build a tree of CMMs. This will not be possible if always considering whole metamodels and their relations, but can be possible if independent concepts are extracted to be treated individually. For that, we will apply our findings on consistency relation decomposition (see Section 3.3). Additionally, the approach specifically aims to improve the specification of transformations for descriptive relations. In consequence, it must be combined with transformations expressing the normative relations between CMMs and other metamodels.

4 STATUS AND PLANNED EVALUATION

We base our studies on the work of Kramer [19] and its languages for binary consistency preservation, integrated into the Vitruvius framework[1] [17]. They allow to define unidirectional and imperative, as well as bidirectional and declarative consistency-preserving transformations and a seamless integration of both. Additionally, Kramer defined several transformations for his evaluation, which we recently extended to an extensive set of transformations for component-based, object-oriented software ensuring pairwise consistency between Java code, UML class and component models, and instances of the Palladio Component Model (PCM) [36], an ADL.

We started to identify problems arising from the black-box combination of binary transformations using the above mentioned transformations. We currently develop solution patterns to allow the generic black-box combination of transformations and apply them to that study. We aim to present a classification of occurring problems and our solution pattern catalog at the International Conference on Model Transformations (ICMT) 2019.

We will also use the case study to investigate the decomposition of consistency relations. For example, on a top level, functional concepts shared between PCM and Java have to be seperated from structural concepts shared between the ADLs and the OO metamodels, which can further be divided into different OO concepts shared between programming languages. Regarding our approach for defining CMMs, we developed a prototypical language that supports their definition [11]. It serves as a proof-of-concept with limited expressiveness, but has already shown the applicability to exemplary relations of the four metamodels of the above mentioned study. After extending the approach and finalizing the language, we aim to apply it to the decomposed relations of the study. This allows us to get initial results on the general applicability of the approaches for decomposition and CMM definitions. We plan to present our results on that at MODELS conference 2019.

To evaluate the generality of our approaches, the pattern catalog, the decomposition concept and CMMs, we will conduct case studies on two additional application domains: embedded automotive software [30, 31], using the metamodels SysML [33], Amalthea [45] and a simplified version of ASCET [10], and production system development [1], based on the AutomationML [13] metamodels CAEX, PLCOpen and Collada. We will perform different kinds of evaluation [5, p. 15]: We will analyze the *applicability* by conducting usage scenarios that we will extract from the histories of existing projects to evaluate if the approaches produce valid and consistent model states. To show the *benefit* of using CMMs with decomposed consistency relations, we will compare our baseline, the results of the combination of binary transformation for investigating transformation interoperability, with those of the CMM approach.

5 CONCLUSION

In this paper, we have motivated the necessity to research multi-model consistency preservation based on the combination of binary model transformations. We identified two essential problems arising from transformation combination: *interoperability* and *specification trade-offs*. In our proposed thesis, we will contribute to theory by identifying common interoperability issues and by developing systematic solutions to ensure general interoperability of transformations. We will also discuss challenges to deal with when combining binary transformations and provide an approach for reducing the necessity to develop trade-off solutions regarding them. This will be based on the decomposition of consistency relations and the explicit specification of common concepts.

We will contribute to practice by delivering a pattern catalog for achieving transformation interoperability that can be used by transformation developers as well as language developers. We will also come up with a specialized language for specifying concept metamodels (CMMs) to make consistency relations explicit. Finally, we will deliver transformations for the consistency relations of the metamodels from our evaluation case studies, which can be readily applied in practical projects.

[1] http://vitruv.tools

REFERENCES

[1] Sofia Ananieva, Erik Burger, and Christian Stier. 2018. *Model-driven consistency preservation in automationml*. In *14th IEEE International Conference on Automation Science and Engineering*. accepted, to appear.

[2] Anthony Anjorin. 2014. *Synchronization of Models on Different Abstraction Levels using Triple Graph Grammars*. Ph.D. Dissertation. Technische Universität Darmstadt.

[3] Anthony Anjorin, Sebastian Rose, Frederik Deckwerth, and Andy Schürr. 2014. "Efficient model synchronization with view triple graph grammars." In *Modelling Foundations and Applications*. LNCS. Vol. 8569. Springer International Publishing, 1–17.

[4] Gábor Bergmann, István Dávid, Ábel Hegedüs, Ákos Horváth, István Ráth, Zoltán Ujhelyi, and Dániel Varró. 2015. "Viatra 3: a reactive model transformation platform." In *Theory and Practice of Model Transformations*. Springer, 101–110.

[5] Rainer Böhme and Ralf Reussner. 2008. "Validation of Predictions with Measurements." In *Dependability Metrics*. LNCS. Vol. 4909. Springer-Verlag Berlin Heidelberg. Part 3, 14–18.

[6] Antonio Cicchetti, Davide Di Ruscio, and Romina Eramo. 2006. *Towards Propagation of Changes by Model Approximations*. In *2006 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06)*. IEEE Computer Society, 24.

[7] Romina Eramo, Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Alfonso Pierantonio. 2012. "A model-driven approach to automate the propagation of changes among Architecture Description Languages." *Software and Systems Modeling*, 11, 29–53, 1.

[8] Romina Eramo, Alfonso Pierantonio, José Raúl Romero, and Antonio Vallecillo. 2008. *Change Management in Multi-Viewpoint System Using ASP*. In *Enterprise Distributed Object Computing Conference Workshops, 2008 12th*, 433–440.

[9] Romina Eramo, Alfonso Pierantonio, and Gianni Rosa. 2015. *Managing Uncertainty in Bidirectional Model Transformations*. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering* (SLE 2015). ACM, 49–58.

[10] Ulrich Freund. 2010. *Representation of Automotive Software Description Means in ASCET*. In *Proceedings of the 2007 International Dagstuhl Conference on Model-based Engineering of Embedded Real-time Systems* (MBEERTS'07). Springer-Verlag, 355–360.

[11] Joshua Gleitze. 2017. *A Declarative Language for Preserving Consistency of Multiple Models*. Bachelor's Thesis. Karlsruhe Institute of Technology (KIT).

[12] Maria-Eugenia Iacob, Maarten W. A. Steen, and Lex Heerink. 2008. *Reusable Model Transformation Patterns*. In *2008 12th Enterprise Distributed Object Computing Conference Workshops*, 1–10.

[13] International Electrotechnical Commission (IEC). 2018. *IEC 62714-1:2018 – Engineering data exchange format for use in industrial automation systems engineering - Automation Markup Language*.

[14] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. 2006. *ATL: A QVT-like Transformation Language*. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications* (OOPSLA '06). ACM, 719–720.

[15] Heiko Klare. 2016. *Designing a Change-Driven Language for Model Consistency Repair Routines*. Master's Thesis. Karlsruhe Institute of Technology (KIT).

[16] Dimitrios Kolovos, Richard Paige, and Fiona Polack. 2008. *Detecting and Repairing Inconsistencies across Heterogeneous Models*. In *2008 1st International Conference on Software Testing, Verification, and Validation*, 356–364.

[17] Max E. Kramer, Erik Burger, and Michael Langhammer. 2013. *View-Centric Engineering with Synchronized Heterogeneous Models*. In *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling* (VAO '13) Article 5. ACM, 5:1–5:6.

[18] Max E. Kramer, Michael Langhammer, Dominik Messinger, Stephan Seifermann, and Erik Burger. 2015. *Change-driven consistency for component code, architectural models, and contracts*. In *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering* (CBSE '15). ACM, 21–26.

[19] Max Emanuel Kramer. 2017. *Specification Languages for Preserving Consistency between Models of Different Languages*. Ph.D. Dissertation. Karlsruhe Institute of Technology (KIT). 278 pp.

[20] Angelika Kusel, Juergen Etzlstorfer, Elisabeth Kapsammer, Philip Langer, Werner Retschitzegger, Johannes Schoenboeck, Wieland Schwinger, and Manuel Wimmer. 2013. *A Survey on Incremental Model Transformation Approaches*. In *ME 2013 – Models and Evolution Workshop Proceedings*, 4–13.

[21] Michael Langhammer. 2017. *Automated Coevolution of Source Code and Software Architecture Models*. Ph.D. Dissertation. Karlsruhe Institute of Technology (KIT). 259 pp.

[22] Michael Langhammer and Max E. Kramer. 2014. "Determining the Intent of Code Changes to Sustain Attached Model Information During Code Evolution." In *Fachgruppenbericht des 2. Workshops "Modellbasierte und Modellgetriebene Softwaremodernisierung"*. Softwaretechnik-Trends. Vol. 34 (2). GI e.V.

[23] Kevin Lano and Shekoufeh Kolahdouz-Rahimi. 2014. "Model-Transformation Design Patterns." *IEEE Transactions on Software Engineering*, 40, 12, 1224–1259.

[24] Kevin Lano, Shekoufeh Kolahdouz-Rahimi, Sobhan Yassipour-Tehrani, and Mohammadreza Sharbaf. 2018. "A Survey of Model Transformation Design Pattern Usage." *Journal of Systems and Software*, 140, 48–73.

[25] Nuno Macedo and Alcino Cunha. 2013. "Implementing QVT-R Bidirectional Model Transformations Using Alloy." In *Fundamental Approaches to Software Engineering*. LNCS. Vol. 7793. Springer Berlin Heidelberg, 297–311.

[26] Nuno Macedo, Alcino Cunha, and Hugo Pacheco. 2014. *Towards a Framework for Multidirectional Model Transformations*. In *3rd International Workshop on Bidirectional Transformations - BX*. Vol. 1133. CEUR-WS.org. CEUR-WS.org.

[27] Nuno Macedo, Tiago Guimaraes, and Alcino Cunha. 2013. *Model repair and transformation with echo*. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, 694–697.

[28] Nuno Macedo, Tiago Jorge, and Alcino Cunha. 2017. "A Feature-based Classification of Model Repair Approaches." *IEEE Transactions on Software Engineering*, PP, 99, 1.

[29] Ivano Malavolta, Henry Muccini, Patrizio Pelliccione, and Damien A. Tamburri. 2010. "Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies." *IEEE Transactions of Software Engineering*, 36, 1, 119–140.

[30] Manar Mazkatli. 2016. *Consistency Preservation in the Development Process of Automotive Software*. Master's thesis. Karlsruhe Institute of Technology (KIT).

[31] Manar Mazkatli, Erik Burger, Anne Koziolek, and Ralf H. Reussner. 2017. *Automotive Systems Modelling with Vitruvius*. In *15. Workshop Automotive Software Engineering* (Lecture Notes in Informatics (LNI)). (Chemnitz). Vol. P-275. GI, Bonn, 1487–1498.

[32] Object Management Group (OMG). 2016. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification – Version 1.3*.

[33] Object Management Group (OMG). 2017. *OMG Systems Modeling Language – Version 1.5*. (2017).

[34] Jon Oldevik. 2005. *Transformation Composition Modelling Framework*. In *Distributed Applications and Interoperable Systems*. Springer Berlin Heidelberg, 108–114.

[35] Maksym Petrenko, Václav Rajlich, and Radu Vanciu. 2008. *Partial Domain Comprehension in Software Evolution and Maintenance*. In *2008 16th IEEE International Conference on Program Comprehension*, 13–22.

[36] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolek, Heiko Koziolek, Max Kramer, and Klaus Krogmann. 2016. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press. 408 pp.

[37] Jesús Sánchez Cuadrado and Jesús García Molina. 2008. *Approaches for Model Transformation Reuse: Factorization and Composition*. In *Theory and Practice of Model Transformations*. Springer Berlin Heidelberg, 168–182.

[38] Thomas Stahl and Markus Völter. 2006. *Model-Driven Software Development*. John Wiley & Sons.

[39] Perdita Stevens. 2017. *Bidirectional Transformations in the Large*. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 1–11.

[40] Frank Trollmann and Sahin Albayrak. 2016. *Extending Model Synchronization Results from Triple Graph Grammars to Multiple Models*. In *Theory and Practice of Model Transformations: 9th International Conference, ICMT 2016, Held as Part of STAF 2016, Vienna, Austria, July 4-5, 2016, Proceedings*. Springer International Publishing, 91–106.

[41] Frank Trollmann and Sahin Albayrak. 2015. *Extending Model to Model Transformation Results from Triple Graph Grammars to Multiple Models*. In *Theory and Practice of Model Transformations: 8th International Conference, ICMT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-21, 2015. Proceedings*. Springer International Publishing, 214–229.

[42] Dennis Wagelaar. 2008. *Composition Techniques for Rule-Based Model Transformation Languages*. In *Theory and Practice of Model Transformations*. Springer Berlin Heidelberg, 152–167.

[43] Dennis Wagelaar, Massimo Tisi, Jordi Cabot, and Frédéric Jouault. 2011. *Towards a General Composition Semantics for Rule-Based Model Transformation*. In *Model Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, 623–637.

[44] Dennis Wagelaar, Ragnhild Van Der Straeten, and Dirk Deridder. 2010. "Module superimposition: a composition technique for rule-based model transformation languages." *Software & Systems Modeling*, 9, 3, 285–309.

[45] Carsten Wolff, Lukas Krawczyk, Robert Höttger, Christopger Brink, Uwe Lauschner, Daniel Fruhner, Erik Kamsties, and Burkhard Igel. 2015. *AMALTHEA – Tailoring tools to projects in automotive software development*. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*. Vol. 2, 515–520.

[46] Yingfei Xiong, Dongxi Liu, Zhenjiang Hu, Haiyan Zhao, Masato Takeichi, and Hong Mei. 2007. *Towards Automatic Model Synchronization from Model Transformations*. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (ASE '07). ACM, 164–173.