# Difference-based Conformance Checking for Ecore Metamodels

Erik Burger, Aleksandar Toshovski

Institute for Programme Structures and Data Organization
Karlsruhe Institute of Technology
Am Fasanengarten 5
76131 Karlsruhe, Germany
burger@kit.edu, aleksandar.toshovski@student.kit.edu

**Abstract:** During modern model-driven development processes, generators and higher-order transformations are used to create metamodels with short life cycles. Since these metamodels often differ from each other only in small parts, instances as well as metamodels may be re-used if the difference between them does not lead to a violation of instance conformance. Existing co-evolution approaches describe this conformance based on change operators to a metamodel. Thus, they require that changes to the metamodels are carried out using special editors. To use this conformance for arbitrarily generated metamodels, we present a conformance validator for Ecore metamodels that is based on difference-based analysis. The validator has been implemented as a plug-in for the Eclipse framework. We demonstrate the completeness of our approach by covering state-of-the-art co-evolution change operators.

## 1 Introduction

In model-driven engineering, instances of metamodels represent entitites in the domain of interest and are thus the primary artefacts which are modified during development of a system. Metamodels, however, represent standards, such as UML, which are implemented in specific modeling tools. Thus, metamodels usually stay stable during the development process. When such a standard or tool evolves, a new version of a metamodel is issued, and existing instances have to be migrated to valid instances of the new versions of the metamodels. Since these evolution steps do not occur frequently and may affect a large number of instances, migration scripts can be provided to adapt those instances. Metamodel evolution mechanisms for automatic co-evolution of instances [BG10; HVW11] support a semi-automatical migration process from one metamodel to another.

In advanced model-driven approaches, such as multi-view modeling [ASB10; Bur+13; Bur13], metamodels, model-to-model transformations, and instances are generated on-the-fly based on declarative definitions. Incremental changes to these definition lead to incremental evolution of the generated metamodels. Thus, the life-cycles for metamodels are considerably shorter. It is, however, desirable to re-use metamodels in such processes for reasons of compatibility to existing tools, and for the development of graphical editors.

In this paper, we present a conformance relation for Ecore metamodels which expresses that instances of one metamodel are also valid instances of another metamodel. This conformance can be used in two ways: First, to determine if co-evolution efforts are necessary for existing instances of a metamodel, and second, to determine whether existing metamodels can be re-used in scenarios where metamodels are generated automatically. In contrast to existing co-evolution methods [Bec+07; Wac07], which require a manual tracking of edit operations, the conformance relation presented in this paper can be determined by a difference-based analysis of two distinct metamodels or two versions of a metamodel. The contribution of this paper is the definition of conformance as a set of rules using the Java Drools[1] rule engine, and a prototypical implementation based on EMF Compare [BP08]. To evaluate the completeness of our approach, we show that the approach covers all operators of the catalogue presented by Herrmannsdörfer in [HVW11]. We demonstrate the application with an extension of the ModelJoin tool [Bur+13].

The rest of this paper is structured as follows: In section 2, we present the concept of the conformance relation, followed by the technical realization in section 3. We evaluate the conformance validation with the operator catalogue of Hermannsdörfer and a modelling example in section 4. We conclude with a brief discussion of related work and prospects on future work in section 5.

## 2 Concept

Incremental changes to metamodels do not necessarily break the compatibility to existing instances, for example, if only additive changes are applied to the metamodels. Metamodels can thus be re-used for multiple instances. This is especially beneficiary if graphical representations and editors have been defined for a specific metamodel, since these would have to be adapted as well otherwise. To exploit the compatibility of existing instances to new versions of a metamodel, we define a conformance relation between metamodels:

**Definition 1 (Conformance)** *Let $M_A$, $M_B$ be metamodels and $I(M_A)$, $I(M_B)$ the sets of all possible instances of $M_A$ and $M_B$. Metamodel conformance is defined as*

$$conforms(M_A, M_B) \Leftrightarrow I(M_A) \subseteq I(M_B)$$

To determine the conformance relation between two actual metamodels, we categorize metamodel changes based on [BG10; HVW11]. These approaches describe the impact of single or multiple changes to a MOF-based metamodel on existing instances. In these terms, conformance of metamodels means that all changes that have to be applied to $M_A$ in order to acquire $M_B$ are *model-preserving* [HVW11] / *non-breaking* [BG10]. The co-evolution approaches mentioned above are, however, operator-based, i.e., they assume that a change between two metamodels is expressed as a series of atomic changes. Edapt [HVW11], for example, requires that the user expresses changes to metamodels as specific refactoring steps using an Eclipse plug-in. If a metamodel is changed by any other than the Edapt

[1] `http://www.jboss.org/drools/`

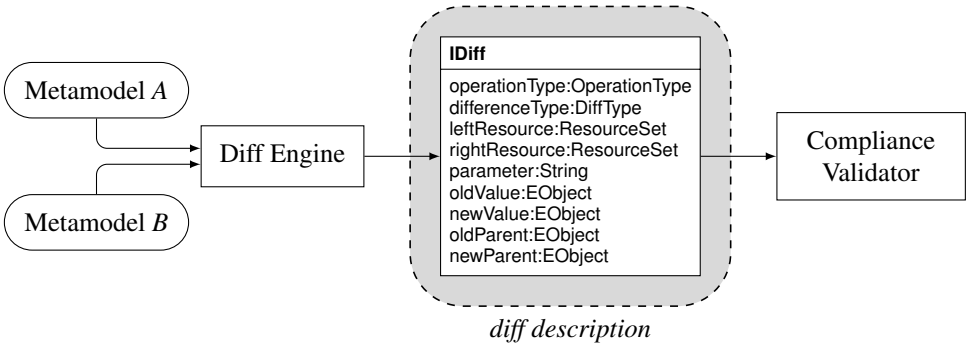


Figure 1: Concept for Determining the Conformance

editor, or generated by a declarative definition as in the example above, the approach is not applicable.

To enable conformance checking between arbitrary metamodels independently of the tools with which they were created, we present a *difference-based* approach for conformance checking. The approach is displayed in Figure 1: A *Diff Engine* is used to determine the diff between two existing metamodels. The calculated `IDiff` element serves as an input for the *Compliance Validator*, which is based on a rule set that covers all possible changes to a metamodel.

## 3 Technical Realization

We have implemented a prototypical *Compliance Validator* which checks if the conformance relation holds for two Ecore metamodels. The architecture of the approach is displayed in Figure 2. The main component *Compliance Validator* contains the logic for checking the conformance relation of Definition 1. It has been implemented as an Eclipse plug-in. Metamodels are persisted in a *Metamodel Repository*, which is used by the validator to retrieve metamodels. The usage of this repository makes it possible to compare a metamodel with several existing metamodels and to find a conforming metamodel in the repository. We have implemented a simple file-based metamodel repository for this purpose. The *Diff Engine* is used to determine the delta between two metamodels. We use EMF Compare for

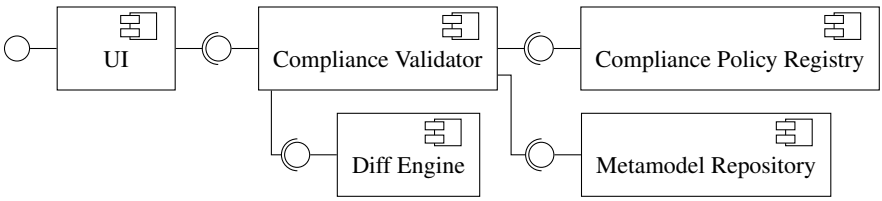


Figure 2: Component model for the Compliance Validator

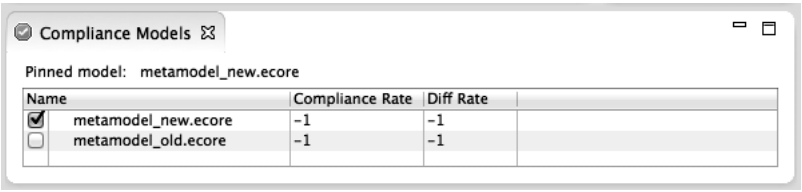


Figure 3: Compliance and Diff Rate in an Eclipse UI view (showing default value −1)

this purpose. The policies for determining the conformity of two metamodels are saved in the *Compliance Policy Registry* component, so users can adapt them and register custom policies. We use the Rete-based Drools rule engine for the definition of the conformance policies. The conformance check is implemented in Eclipse via a *UI* component.

We have analysed all possible changes to Ecore metamodels, based on the classification of metamodel changes in [BG10] and [HVW11]. Since these works are based on MOF, adaptations were necessary to take the differences between MOF and Ecore into account. For each change type, we created a rule which describes whether a change type violates the conformity of existing instance to the metamodel which is being changed. In total, we have defined 24 rules which cover all model-preserving change types.

An example for a conformance rule is displayed in Listing 1: The rule analyses the impact of the deletion of a structural feature from an `EClass` element. The `IDiff` element describes the delta between two elements of the respective metamodels. The Java helper functions `isPullUpFeature()` and `isParentAbstract()` in the `DroolsUtils` library determine whether the class in the old metamodel is abstract and whether the feature was moved to a superclass, which influences the impact of the change. The `then`-clause of the rule is empty since we use a listener to react on the firing of a rule.

```
rule "ReferenceChange␣EClass␣remove␣Attribute/Reference"
   when diff: IDiff( operationType == OperationType.DELETE, differenceType ==
        DiffType.REFERENCE, parameter=="eStructuralFeatures", DroolsUtils.
        isPullUpFeature(oldValue,newValue,newParent)
   then
end
```

Listing 1: Drools rule for deletion of an attribute/reference

The complete set of rules is not represented in this paper due to space restrictions. We refer to the reader to [Tos13][2] for an extensive description of the rules and the validator.

If there are several metamodels in the Metamodel Repository which conform to the metamodel which is being checked, a measure for the similarity of metamodels is calculated to determine the metamodel with the lowest number of conflicts (see Figure 3): The *compliance rate* is the number of changes which violate conformance, while the *diff rate* describes the number of total changes. The user of the validator is furthermore presented a list of issues which cause the inconformance, and can adapt the metamodel accordingly to re-validate it.

[2] `http://sdqweb.ipd.kit.edu/publications/pdfs/toshovski2013a.pdf` (in German)

# 4 Evaluation

Although the conformance relation (Defintion 1) is formally defined, we consider it impractical to formally prove the correctness of our implementation, since the Ecore metamodel and MOF itself lack a formal basis which would be necessary for such a proof. Thus, we follow the same approach as Herrmannsdörfer in [HVW11] and demonstrate the practical completeness of our implementation by validating the coverage of the most frequent cases of metamodel changes in practice.

Since the purpose of our conformance relation is twofold, we will evaluate two cases: First, we will show that our conformance validator covers all the operations in the catalog [HVW11], thus demonstrating that the conformance validation can be used for co-evolution scenarios. Second, we will demonstrate the applicability of the conformance validator for the re-use of metamodels in cases where instances and metamodels are generated from a declarative definition. To this end, we have integrated the approach with the ModelJoin tool [Bur+13] and checked the conformance of changes using a joined metamodel based on the Palladio Component Model [BKR09] and a metamodel for simulation results.

## 4.1 Change Operators by Herrmannsdörfer et al.

In their 2011 publication [HVW11], Hermannsdörfer et al. have presented a catalog of operators for the coupled evolution of metamodels and models, which covers common cases of metamodel adaptations. The operators are divided into three groups: *structural primitives*, *none-structual primitives*, and *complex operations*. Each operator is classified by the impact it has on existing instances. For our conformance relation, the class of *model-preserving* operators is of interest, since it describes the cases where no adaptation to existing instances is necessary. The group of primitive operators can be described by single instances of the `IDiff` element, while complex operators have to be describe as a set of `IDiff` elements.

To evaluate the completeness of the compliance validator, we wrote a JUnit test suite that applied each of the change operations to an example metamodel and tested whether the validator was able to detect the correct class of changes. For primitive operations, we created the appropriate `IDiff` elements directly; for complex changes, we used Edapt to apply the change to the example metamodel. The rule set of the conformance validator was able to detect all the 61 operations of the catalog correctly.

## 4.2 ModelJoin views on the Palladio Component Model

The ModelJoin [Bur+13] tool generates custom metamodels and instances based on textual queries (see Figure 4): Based on a query, an annotated target metamodel is synthesized and a QVT-O transformation is generated automatically based on the annotations in the target metamodel. Since every execution of a query leads to the generation of a query-specific target metamodel, the re-usability of query results is limited if metamodel-specific

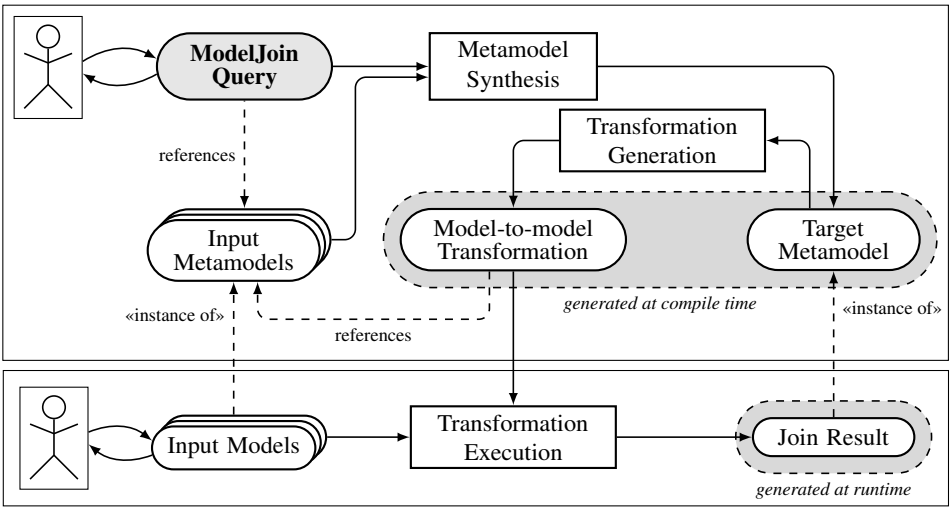


Figure 4: Metamodel and transformation generation in ModelJoin (in FMC [KGT06] notation, from [Bur+13])

visualisations or further transformations are used. To enable the re-use of metamodels, we have extended the ModelJoin tool by the conformance validator and the file-based metamodel repository. If a query is executed, the conformance validator checks if there are metamodels in the repository to which the newly synthesized metamodel conforms. If one or more suitable metamodels are found, the user can choose one of these, so the join result is generated as an instance of this metamodel. If no suitable metamodel is found, the user can access the list of incompatible changes and either modify the query accordingly until the synthesized metamodel conforms to one of the metamodels in the repository, or use the synthesized metamodel, which is then added to the repository.

We have tested the compliance validator by varying ModelJoin queries on the Palladio Component Model [BKR09] and the Sensor Framework metamodel, checking the conformance against formerly created metamodels. Although these tests are not extensive enough to give information about the percentage of cases in which metamodels can be re-used, our first experiences indicate that additive changes profit from the conformance check. If a query is amended with additional properties of the source metamodels, former queries and their instances can still be used with the newly generated metamodel. This way, the user gets feedback if additions to a query break the compatibility to queries which are already in use.

## 5 Related Work/Conclusion

*Metamodel evolution* adresses the compliance between different versions of a metamodel. Several approaches support the analyses of metamodel changes by describing the changes

in a model-based format [Bec+07; BG10] and generating migration transformations automatically [Cic+08]. Automatic derivation of this difference description has been proposed in [DIP12] as a base for migration scripts, but without categorization of conformance.

The conformance relation presented in this paper can be seen as a special case of model typing [SJ07]. Our state-based conformance check can be used for model type checking of Ecore metamodels, which determines a subtype-relation between two metamodels.

In this paper, we have presented a conformance relation between Ecore metamodels which expresses that all instances of one metamodel are valid instances of another metamodel. This relation is useful either for the re-use of instances when metamodels evolve, or for the re-use of metamodels in approaches where metamodels and instances are generated from declarative definitions. The contribution of this paper is a difference-based compliance validator which analyses Ecore metamodels for conformance based on a set of rules. The advantage of a difference-based analysis is the independence from metamodelling tools with which the metamodels under study are created.

In contrast to the change impact analysis of [BG10] and the classification of change operators of [HVW11], the conformance validator only detects model-preserving changes and does not provide means for resolving other changes that violate the conformance. In future work, the rule set can be extended to differentiate between conflicts which have to be resolved manually and those which can be fixed automatically.

The conformance validator is limited to Ecore-based metamodels and is thus tied to the EMF framework. We do however not see this as a serious limitation due to the wide acceptance of EMF and Eclipse in industry and research. As future work, the conformance validator can be extended to support other metamodel repositories such as CDO[3], Teneo[4], or EMFStore[5].

## References


[ASB10] Colin Atkinson, Dietmar Stoll, and Philipp Bostan. “Orthographic Software Modeling: A Practical Approach to View-Based Development”. In: *Evaluation of Novel Approaches to Software Engineering*. Ed. by Leszek A. Maciaszek, César González-Pérez, and Stefan Jablonski. Vol. 69. Communications in Computer and Information Science. Berlin/Heidelberg: Springer, 2010, pp. 206–219.

[Bec+07] Steffen Becker et al. “A Process Model and Classification Scheme for Semi-Automatic Meta-Model Evolution”. In: *Proc. 1st Workshop MDD, SOA und IT-Management (MSI’07)*. GiTO-Verlag, 2007, pp. 35–46.

[BG10] Erik Burger and Boris Gruschko. “A Change Metamodel for the Evolution of MOF-Based Metamodels”. In: *Modellierung 2010, Klagenfurt, Austria, March*


[3] `http://www.eclipse.org/cdo`

[4] `http://wiki.eclipse.org/Teneo`

[5] `http://www.eclipse.org/emfstore`

*24-26, 2010*. Ed. by Gregor Engels, Dimitris Karagiannis, and Heinrich C. Mayr. Vol. P-161. GI-LNI. 2010.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. “The Palladio component model for model-driven performance prediction”. In: *Journal of Systems and Software* 82 (2009), pp. 3–22.

[BP08] Cédric Brun and Alfonso Pierantonio. “Model Differences in the Eclipse Modelling Framework”. In: *UPGRADE The European J for the Informatics Professional* IX.2 (2008), pp. 29–34.

[Bur+13] Erik Burger et al. *ModelJoin Technical Report*. 2013. URL: http://sdqweb.ipd.kit.edu/publications/pdfs/burger2013modeljoin.pdf.

[Bur13] Erik Burger. “Flexible Views for View-Based Model-Driven Development”. In: *Proceedings of the 18th international doctoral symposium on Components and architecture*. WCOP ’13. Vancouver, British Columbia, Canada: ACM, 2013, pp. 25–30.

[Cic+08] Antonio Cicchetti et al. “Automating Co-evolution in Model-Driven Engineering”. In: *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*. EDOC ’08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 222–231.

[DIP12] Juri Di Rocco, Ludovico Iovino, and Alfonso Pierantonio. “Bridging state-based differencing and co-evolution”. In: *Proceedings of the 6th International Workshop on Models and Evolution*. ME ’12. Innsbruck, Austria: ACM, 2012, pp. 15–20.

[HVW11] Markus Herrmannsdörfer, Sander D. Vermolen, and Guido Wachsmuth. “An extensive catalog of operators for the coupled evolution of metamodels and models”. In: *Proceedings of the Third international conference on Software language engineering*. SLE’10. Berlin/Heidelberg: Springer, 2011, pp. 163–182.

[KGT06] Andreas Knöpfel, Bernhard Gröne, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Wiley, 2006.

[SJ07] Jim Steel and Jean-Marc Jézéquel. “On model typing”. English. In: *Software & Systems Modeling* 6.4 (2007), pp. 401–413.

[Tos13] Aleksandar Toshovski. “Wiederverwendung von Metamodellen in ModelJoin-Sichten”. MA thesis. Am Fasanengarten 5, 76131 Karlsruhe, Germany: Karlsruhe Institute of Technology (KIT), July 2013.

[Wac07] Guido Wachsmuth. “Metamodel Adaptation and Model Co-adaptation”. In: *ECOOP 2007 – Object-Oriented Programming*. Ed. by Erik Ernst. Vol. 4609. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 600–624.