A Process Model and Classification Scheme for Semi-Automatic Meta-Model Evolution

Steffen Becker, Thomas Goldschmidt, Boris Gruschko, Heiko Koziolek
Chair Software Design & Quality, University of Karlsruhe
FZI Forschungszentrum Informatik, SAP Research
Graduiertenkolleg Trustsoft,[*] University of Oldenburg
{sbecker,goldschm,gruschko,koziolek}@ipd.uka.de

Abstract: Model Driven Software Development (MDSD) has matured over the last few years and is now becoming an established technology. As a consequence, dealing with evolving meta-models and the necessary migration activities of instances of this meta-model is becoming increasingly important. Approaches from database schema migration tackle a similar problem, but cannot be adapted easily to MDSD. This paper presents work towards a solution in the model-driven domain. Firstly, we introduce a process model, which defines the necessary steps to migrate model instances upon an evolving meta-model. Secondly, we have created an initial classification of meta-model changes in EMF/Ecore utilised by our process model.

1 Introduction

Since the introduction of the Model Driven Architecture (MDA) ideas several years ago, methods and tools have matured in a significant way [VS06]. This resulted in a more widespread application of model-driven techniques in industry as well as in academia. However, with the increased application of these techniques new challenges arise, which hinder applicability of MDA in large-scale industrial projects.

A challenge are evolving meta-models, which are used to build domain-specific languages and transformations in MDA. Changing requirements (often resulting in a new version of the software system) or refactorings during the software design process, also lead to evolving meta-models. With respect to meta-models, often a better applicability and understandability of the resulting domain-specific language is an important driving force for these kinds of refactorings. However, changing a meta-model often results in invalid model instances, which have to be migrated.

There are only few existing approaches tuned specifically to the needs of commonly used meta-meta-models, like Ecore or MOF. Most of the related work comes from the area of database schema evolution. Because of different expression powers in Entity-Relationship modelling and Object-Oriented modelling languages, a simple transfer of existing approaches is hard to achieve.

[*]This work is supported by the German Research Foundation (DFG), grant GRK 1076/1

The contributions of this paper are (a) a process model for migrating model instances upon meta-model changes semi-automatically and (b) an initial classification scheme for meta-model changes in the Eclipse Modelling Framework (EMF)[BSM$^+$03]. Part of this classification is a statement whether the respective model-change can be resolved automatically or semi-automatically.

The structure of this paper is as follows. First, the necessary foundations are introduced briefly. In Section 3, the problem is illustrated using a small example. Section 4 introduces our process model. A classification schema of meta-model changes found in instances of the Ecore meta-meta-model is presented in Section 5. This paper finishes with a brief overview on related work and a concluding summary as well as future work.

2 Foundation

The UML [OMG05] is a modelling language for object-oriented software describing artefacts in some domain of interest to the user. Early versions of the UML relied on textual descriptions to define the syntax and semantics of the language. In order to standardise and formalise the description of modelling languages such as the UML, the concept of meta-models has been applied. The Object Management Group (OMG) introduced the Meta Object Facility (MOF) [OMG06] as a standard for describing such meta-models. Models on this level are therefore called meta-meta-models. Hence, the result is a four-layered architecture.

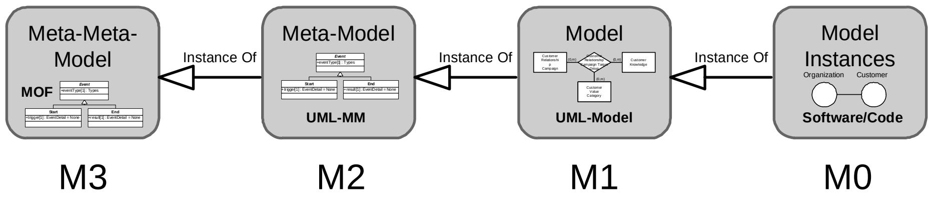


Figure 1: MOF Layers

Fig. 1 shows a typical example for this four-layered architecture. The M0 layer consists of actual data (in the case of UML this would be the software that is developed using the UML model), M1 contains models that describe the data (e.g. a UML model of a software system), M2 contains meta-models that describe how models can be built (e.g. the UML meta-model itself) and finally the M3 layer with the meta-meta-model (in our example this would be the MOF).

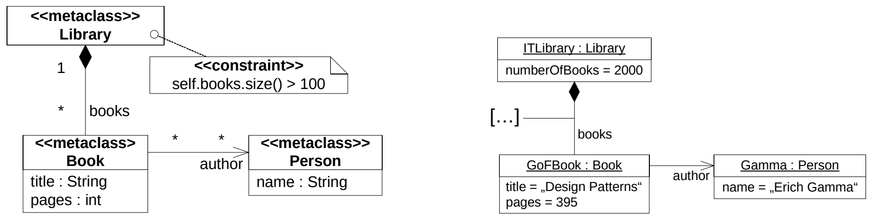
Version 2.0 of the MOF [OMG06] includes two different variants: the Essential MOF (EMOF) and the Complete MOF (CMOF). EMOF specifies basic constructs such as packages, classes and properties. CMOF adds concepts such as associations as first class entities, exceptions, element and package imports. EMF defines a meta-meta-model called

Ecore, which is aligned to EMOF. Neither MOF nor Ecore define strategies on how to handle M1 elements when their meta-model (M2) has changed.

In this paper, we focus on Ecore as meta-meta-model (M3). We plan to extend it to the Complete MOF.

3 The Challenge of Model Evolution

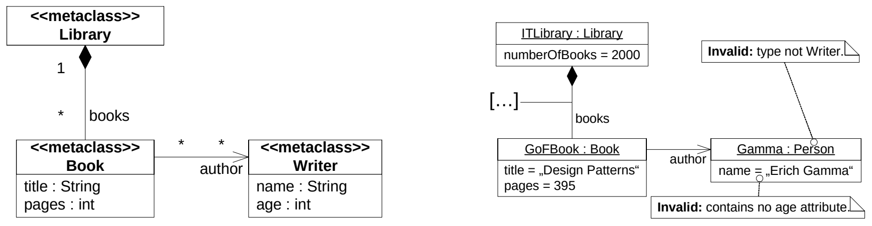
In the following, we illustrate the problem of model evolution with a simple example. Fig. 2(a) depicts a meta-model for libraries. A library contains a number of books, which have been authored by a number of persons. To be considered as an actual library, each library should contain at least 100 books. Fig. 2(b) shows a valid instance of this meta-model (with only one book shown here).



(a) (b)

Figure 2: Example Meta-Model and Corresponding Instance

Once the meta-model is revised, already existing model instances become invalid. Fig. 3(a) shows a modified version of the meta-model with three changes: the class `Person` has been renamed to `Writer`, the renamed class contains a new attribute `age`, and libraries do not have to contain at least 100 books anymore. The model instance in Fig.3(b) is not valid according to the modified meta-model.



(a) (b)

Figure 3: Modified Meta-Model and Invalid Model Instance

The three modifications represent the three types of meta-model changes we consider.

Renaming a meta-model class is a *breaking & resolvable* change. A tool can rename the classes in all known model instances without the need for user input. Introducing a new attribute is a *breaking & not resolvable* modification. When the attribute is added, its values in existing instances cannot be determined. When the change is introduced, a default value could be provided. However, developers of model instances should consider the default value's correctness when migrating their instances. The removed constraint is an *additive* or *non-breaking* change and does not invalidate any existing model instance. It only increases the number of possible instances.

4 The Approach

In the following, we introduce a process model for semi-automatic model instances migration, in face of meta-model changes. Fig. 4 provides an overview of the envisioned process. The presented approach is tailored to minimize the manual effort required to migrate M1 models. As input, the presented process expects either two revisions of a M2 model, or an older version of the M2 model and a trace of performed changes. Furthermore, a set of instances of the older M2 model version is needed. After process completion, the migration process delivers a set of valid M2 model instances.

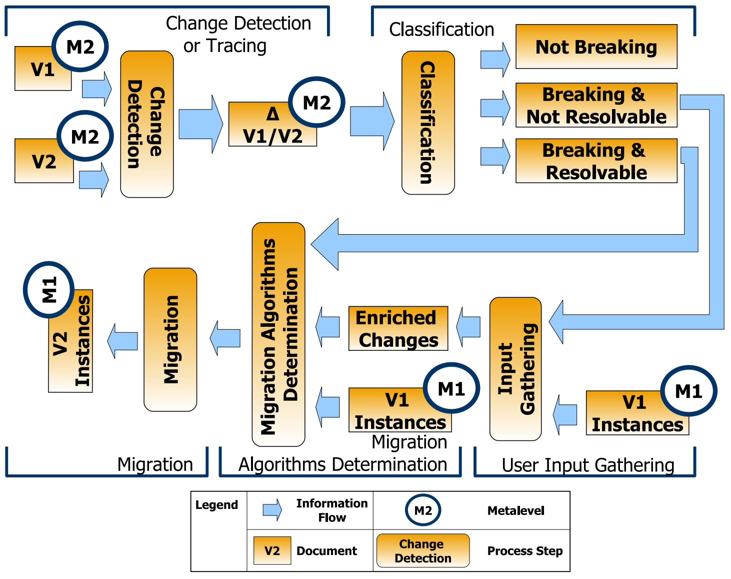


Figure 4: Proposed migration process model.

The following steps have to be performed: change detection (Sec. 4.1), migration algorithm construction (Sec. 4.2), and migration algorithm execution (Sec. 4.3). The change detection step provides the differences between the M2 model revisions to the subsequent step. The differences are represented as an instance of a Change M2 Model specific to the M3 model, in this case Ecore. During migration algorithm construction, the changes are classified according to the scheme presented in Sec. 3. Furthermore, the partitions of the change set are being delivered, to subsequent steps. Finally, the constructed migration algorithm is executed, resulting in a successful process outcome.

4.1 Change Detection

The differences between M2 model revisions can be obtained by applying two different methods. The first method is the *direct comparison* of the revisions, without taking into account any other information. This enables change detection disregarding the tools used for meta-model revisioning. However, this approach would require considerable effort in construction of the comparison algorithms. This algorithm is expected to have high complexity with regard to memory allocation and computation time. Another drawback of the direct comparison, is its dependence on availability of stable model element identifiers. In the absence of stable unique identifiers, the direct comparison would not be capable to detect element moves and would detect a deletion followed by an addition instead.

Another approach, delivering the set of changes performed on the M2 model, is the recording of the *change trace* while the changes are being performed by the M2 model author. The obvious drawback is the need to modify the modelling tools, to enable trace recording. The benefit of the changes trace recording is its capability to detect changes (like element moves) in absence of unique element identifiers.

As output, the differences detection stage produces an instance of a *Change M2 model* with links to the older version of the M2 model.An example of a Change M2 model is the EMF Change meta-model. This Change meta-model describes instance changes of any Ecore based M2 model. However, this M2 model is considered to be too general for our goal. We propose refining it, in order to describe the changes in terms specific to a fixed M3 model (i.e., Ecore). For example, the instance of `FeatureChange`, from EMF Change meta-model, with link to the `name` attribute of a `Ecore.EStructuralFeature` instance, would become an instance of *StructuralFeatureRename* class. This refinement would allow for simplification in the following migration stages, because the interpretation step could be omitted from them.

4.2 Migration Algorithm Construction

The construction of the algorithm for M1 model migration can be divided into the following steps: (a) classification of the detected changes, (b) user input gathering for changes requiring human assistance, and (c) determination of appropriate algorithms for the M1 model migration.

The classification of the detected changes is needed in order to partition the set of detected changes into additive (non-breaking), breaking and resolvable, and breaking and not resolvable changes. The additive changes do not have an effect on the validity of the M1 models. An example of an additive change is the addition of an M2 class without further references and constraints. The breaking and resolvable changes invalidate existing M1 models, but can be resolved by automatic means. An attribute rename would be a change in this category. The breaking but not resolvable changes require human assistance, to allow for the M1 model migration. For example, splitting an M2 class into two classes would invalidate M1 models and require manual intervention to reclassify the existing M1 instances.

After classification of the changes user input required to migrate the M1 models affected by the not resolvable changes. To minimize the effort required to create this user input, an assistance framework will be created, providing action templates for the most common cases.

The algorithms for the M1 model migration can range from a simple name substitution via a lookup table for feature renames, to a full fledged model transformation for the not resolvable cases. In the scope of our work, we envision a library of pre-configurable algorithms, for the most common cases and platform extensions for declarative model-to-model transformations to deal with not resolvable cases. The output of this process step is expected to be an executable program, capable of transforming M1 instances of the older M2 model revision into valid M1 instances of the newer M2 model revision.

4.3 Migration Algorithm Execution

It is possible to execute the migration during M1 model instantiation. We call this option *lazy* migration. Another possibility is migration immediately following M2 model changes. During this so-called *eager* migration, no changes on M1 models are allowed making it an atomic event. The most appropriate point in time for the migration execution is still to be determined.

Early experiments with a vertical prototype, capable of M1 model migration in face of attribute renames, suggest a *lazy* migration. However, this approach may become infeasible for migration of more complex change sets. The appropriate ordering of migration execution may become the most crucial part in the determination of the model life-cycle phase, where the migration is to take place. It is also possible to split the migration into eager and lazy executable parts. This process step is still in an early stage.

5 Classification Scheme Change Operations EMF

We categorized all possible operations on the Ecore M3 model according to the different classes: additive/non-breaking (A), breaking and resolvable (R) and breaking and not resolvable (N). A condensed version of Ecore is depicted in Fig. 5.

We split the possible meta-model changes into to different parts: (a) the addition or removal of elements in the meta-model and (b) changes to attributes of meta-model ele-

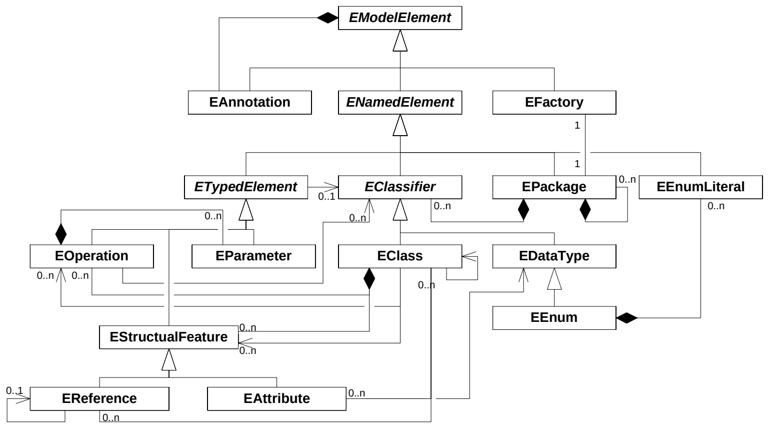


Figure 5: The Ecore meta-meta-model

ments. For each element and attribute in Ecore we identified the impact on instances of the meta-model. Changes resulting from the addition or removal of an element to the meta-model are classified in table 1. The results of attribute changes on any of the meta-model elements are listed in table 2. Elements with no impact on M1 model-elements, e.g. `EOperation` or `EFactory` are not listed in the tables.

Due to lack of space we cannot explain each row of the tables in detail. However, two interesting cases are:

- **Addition of an `EReference`:** Two cases have to be considered, if a new reference is added from a class C_1 to another class C_2. If the reference has a minimum cardinality of 0, M1 models do not break, because instances of C_1 are still allowed to have *no* link to one or more instances of C_2. However, if the minimum cardinality of the new reference is greater than 0 all M1 models that contain instances of C_1 will break. The cardinality constraint is not fulfilled and a link to a C_2 instance cannot be established automatically, because it is unknown to which particular C_2 instance the link should be created.

- **Changing attribute "ordered" from `false` $\Rightarrow$ `true`:** In `ETypedElement`, this is a non-breaking change, because the referred elements have a natural order, i.e., the order in which they are stored. Upon the change, this order is just made explicit. However, this change should yield a warning to inform the user that this order may not be the one he may have expected.

Attr./Ref.	A	R	N	Comment
ENamedElement				
name		✔		
EPackage				

Continued on next page

Attr./Ref.	A	R	N	Comment
nsURI		✔		
nsPrefix		✔		
eClassifiers		✔		
eSubpackages	✔			
eSuperPackage		✔		
ETypedElement				
ordered	✔			false⇒true should yield a warning
unique	✔			true⇒false
			✔	false⇒true
lowerBound	✔			decrease
			✔	increase
upperBound	✔			increase
			✔	decrease
eType	✔			new type super class of old type
			✔	any other case
EStructuralFeature				
changeable	✔			
volatile		✔		true⇒false
		✔		false⇒true and lowerCardinality == 0
			✔	false⇒true and lowerCardinality != 0
transient		✔		true⇒false: compute attr. and store value
		✔		false⇒true: delete all values from M1 model
defaultValueLiteral	✔			
unsettable	✔			
			✔	
derived		✔		same as transient
eContainingClass		✔		
ERefence				
containment		✔		M1 instances not contained in other object
			✔	M1 instances contained in other object
resolveProxies	✔			ignored
eOpposite			✔	
EAttribute				
iD		✔		all values of the structural feature are unique
			✔	non unique values exist
EClass				
abstract	✔			true⇒false
		✔		false⇒true: instances of the class not required
			✔	false⇒true: Instances required: not resolvable
interface	✔	✔	✔	same as abstract
eSuperTypes	✔			added/removed super type empty
	✔			added super type has non-mandatory features
		✔		removed super type contains structural features

Continued on next page

Attr./Ref.	A	R	N	Comment
			✔	added super type contains mandatory features
eStructuralFeatures	✔			added feature not mandatory
		✔		feature removed
			✔	added feature mandatory
eOperations	✔			
EEnum				
eLiterals	✔			literals added
	✔			literals removed, no refering M1 entities
			✔	otherwise
EEnumLiteral				
value				ignored due to atomicity of enum literals
instance				same as value
literal				same as value
eEnum				same as value

Table 2: Classification of attribute changes in Ecore

6 Related Work

The general field of model management has been defined in [BHJ+00] under the term of *Generic Model Management*. This field has been developed by Melnik in [Mel04]. It is our intention to embed our research in the field of Generic Model Management. Further work on the general field of model migration has been performed by Sprinkle in [Spr03]. The classification of M2 model changes and the visualization aspects developed by Sprinkle in [SK04] are of significance to the changes classification and user input gathering stages of our approach.

There are no industrial standards concerning the topic of M1 model migration. The OMG has developed the MOF 2.0 versioning specification [Obj05]. This standard is concerned with attaching versioning information to M2 models, but fails to provide means to obtain the versioning information. The M1 models are not in scope of this standard.

The comparison of M2 models closely resembles the problem of UML model comparison. One algorithm suitable for this comparison is UMLDiff$_{cld}$, proposed by Girschick in [Gir06]. A more general approach to the difference creation between two models is given by Alanen and Porres in [AP03]. These approaches are suitable for the direct comparison option. The changes trace approach is closely related to the problem of information loss in versioning systems, described by Robbes and Lanza in [RL06].

Other related work comes from the areas of model integration and incremental transformation updates. Model integration is concerned with merging multiple models into a single model, e.g. [MSD06]. However, model integration deals with combining models on the same meta-level. The focus of this paper is to support evolution between the model and the meta-model level. Incremental transformations update results of transformations with

Element	Op.	A	R	N	Comment
EPackage	Add	✔			
	Remove		✔		content resolvable
				✔	content unresolvable
EClass	Add	✔			
	Remove		✔		Casc. delete on outgoing containment ref.
EDataType	Add	✔			
	Remove			✔	All attributes typed with the concerned data type have to be converted
EEnum	Add	✔			
	Remove			✔	Same as EDataType
EEnumLiteral	Add	✔			
	Remove			✔	All M1 instances with this value have to be converted to another value
EReference	Add	✔			min. cardinality == 0
				✔	min. cardinality $>$ 0
	Remove		✔		All corresp. M1 links have to be removed
EAttrribute	Add		✔		min. cardinality == 0
				✔	min. cardinality $>$ 0
	Remove		✔		All M1 instances have to be removed

Table 1: Classification of add and remove operations on an Ecore meta-model

a minimum of steps whenever the source model is changed. For this, a change detection is also needed. Hearnden et al. distinguish between the addition of model elements and their removal [HLR06]

7 Conclusions

In this paper, we present an initial approach to tackle the problem of model evolution in MDSD. Our process model aims at a semi-automatic migration of M1 model elements upon M2 model changes and tries to minimize manual interaction. As an initial step, we classified the meta-model operations in the context of the Ecore meta-meta-model.

However, the application of the proposed model migration approach is based on the following assumptions. Firstly, the used M3 model is fixed. Therefore, it is difficult to adopt the approach to M3 model changes. Secondly, capturing of semantical changes is not considered. The only possibility to reflect changes in the semantic of the M2 model, is in the manual migration stage. Finally, the proposed approach depends on the unique identification of M2 model elements to detect their moves.

Extending the approach to another M3 model (e.g., CMOF) is part of future work. Therefore, concepts not included in Ecore, have to be analysed according to scheme presented in Sec. 5. Additional future work is directed at finishing the remaining steps and applying the method in industrial context.

References

[AP03] Marcus Alanen and Ivan Porres. Differences and Union of Models. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *UML 2003 - The Unified Modeling Language. Model Languages and Applications. 6th International Conference, San Francisco, CA, USA, October 2003, Proceedings*, volume 2863 of *LNCS*, pages 2–17. Springer, 2003.

[BHJ+00] Philip A. Bernstein, Laura M. Haas, Matthias Jarke, Erhard Rahm, and Gio Wiederhold. Panel: Is Generic Metadata Management Feasible? In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB*, pages 660–662. Morgan Kaufmann, 2000.

[BSM+03] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy Grose. *Eclipse Modeling Framework*. Addison Wesley Professional, 2003.

[Gir06] Martin Girschick. Difference Detection and Visualization in UML Class Diagrams. Technical Report TUD-CS-2006-5, TU Darmstadt, 2006.

[HLR06] David Hearnden, Michael Lawley, and Kerry Raymond. Incremental Model Transformation for the Evolution of Model-Driven Systems. In *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings*, volume 4199 of *Lecture Notes in Computer Science*, pages 321–335. Springer, 2006.

[Mel04] Sergey Melnik. *Generic Model Management: Concepts and Algorithms*, volume 2967 of *Lecture Notes in Computer Science*. Springer, 2004.

[MSD06] Tom Mens, Ragnhild Van Der Straeten, and Maja D'Hondt. Detecting and Resolving Model Inconsistencies Using Transformation Dependency Analysis. In *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings*, volume 4199 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2006.

[Obj05] Object Management Group. *MOF2 Versioning Final Adopted Specification*, 2005.

[OMG05] OMG Object Management Group. Unified Modeling Language Specification: Version 2, Revised Final Adopted Specification (ptc/05-07-04), 2005.

[OMG06] OMG Object Management Group. MOF 2.0 Core Final Adopted Specification (ptc/03-10-04), 2006.

[RL06] Romain Robbes and Michele Lanza. Change-based software evolution. In *Proceedings of EVOL 2006 (1st International ERCIM Workshop on Challenges in Software Evolution)*, pages 159–164, 2006.

[SK04] Jonathan Sprinkle and Gabor Karsai. A domain-specific visual language for domain model evolution. *J. Vis. Lang. Comput*, 15(3-4):291–307, 2004.

[Spr03] Jonathan Sprinkle. *Metamodel Driven Model Migration*. PhD thesis, Vanderbilt University, Nashville, TN 37203, August 2003.

[VS06] Markus Völter and Thomas Stahl. *Model-Driven Software Development*. Wiley, 2006.