EMF Refactor: Specification and Application of Model Refactorings within the Eclipse Modeling Framework[☆]

Thorsten Arendt[a], Florian Mantz[b], Gabriele Taentzer[a]

[a]*Philipps-Universität Marburg, FB12 - Mathematics and Computer Science, Hans-Meerwein-Strasse, D-35032 Marburg, Germany*
[b]*Høgskolen i Bergen, Department of Computer Engineering, Nygårdsgaten 112, N-5020 Bergen, Norway*

Abstract

Models are the primary artifacts in software development processes following the model-based paradigm. Therefore, software quality and quality assurance frequently leads back to the quality and quality assurance of the involved models. In our approach, we propose a two-phase model quality assurance process considering syntactical model quality based on quality assurance techniques like model metrics, model smells, and model refactorings. In this paper, we present *EMF Refactor*, a new Eclipse incubation project providing specification and application of refactorings wrt. models based on the Eclipse Modeling Framework.

Keywords: modeling, model-based software development, model quality, model refactoring

1. Introduction

The paradigm of model-based software development has become more and more popular, since it promises an increase in the efficiency and quality of software development. It is sensible to address quality issues of artifacts in early software development phases already, for example the quality of the involved models. Especially in model-driven software development, models become primary artifacts where quality assurance of the overall software product considerably relies on the quality assurance of involved software models.

The quality of software models comprises several dimensions. In our approach, we consider a quality assurance process that concentrates on the syntactical dimension of model quality. Syntactical quality aspects are all those which can be checked on the model syntax only. They include of course consistency with the language syntax definition, but also other aspects such as conceptual integrity using the same patterns and principles in similar modeling situations and conformity with modeling conventions often specifically defined for software projects. These and other quality aspects are discussed in [1] for example, where the authors present a taxonomy for software model quality.

Typical quality assurance techniques for models are model metrics and refactorings, see e.g. [2–5]. They originate from corresponding techniques for software code by lifting them to models. Especially class models are closely related to programmed class structures in object-oriented programming languages such as C++ and Java. For behavior models, the relation between models and code is less obvious. Furthermore, the concept of code smells can also be lifted to models leading to model smells. Again code smells for class structures can be easily adapted to models, but smells of behavior models cannot be directly deduced from code smells.

In [6], we present the integration of these techniques in a predefined quality assurance process that can be adapted to specific project needs. Figure 1 shows the two-phase process: Before a software project starts, project- and domain-specific quality checks and refactorings have to be defined. Quality checks are formulated by model smells which can be specified e.g. by model metrics and anti-patterns. Here, the project-specific process can reuse general metrics, smells, and refatorings, as well as special ones specific for the intended modeling purpose. Thereafter, the specified

[☆]This work has been partially funded by Siemens Corporate Technology, Germany.

Email addresses: `arendt@mathematik.uni-marburg.de` (Thorsten Arendt), `fma@hib.no` (Florian Mantz), `taentzer@mathematik.uni-marburg.de` (Gabriele Taentzer)

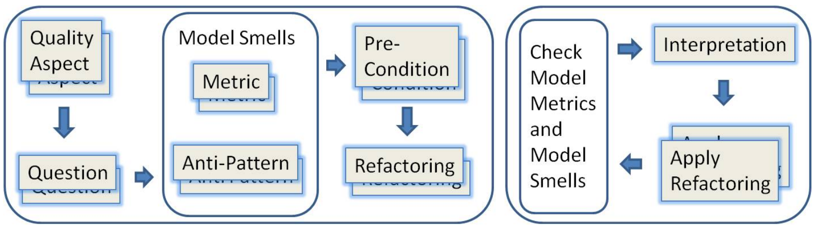


Figure 1: Project-Specific Quality Assurance Process - Specification (left side) and Application (right side)

quality assurance process can be applied to concrete software models by computing model metrics, reporting all model smells and applying model refactorings to erase smells that indicate clear model defects. However, we have to take into account that also new model smells can come in by refactorings. This check-improve cycle should be performed as long as needed to get a reasonable model quality.

Since a manual model review is very time consuming and error prone, it is essential to automate the tasks as effectively as possible. We implemented tools supporting the included techniques metrics, smells, and refactorings for models based on the Eclipse Modeling Framework (EMF) [7], a common open source technology in model-based software development. In this paper, we present the new **Eclipse incubation project *EMF Refactor* [8] supporting specification and application of refactorings wrt. specific EMF based models**.

This paper is organized as follows: In the next section, the model refactoring specification process using *EMF Refactor* is presented. In Section 3, we describe how model refactorings are executed in *EMF Refactor*. Finally, related work is discussed and a conclusion is given.

2. Specification of EMF Model Refactorings

As already mentioned, *EMF Refactor* supports specification and application of refactorings wrt. EMF based models. In the following, we use an example UML2EMF model to demonstrate the capabilities of *EMF Refactor*. Please note, that *EMF Refactor* can be used on arbitrary models whose meta models are instances of EMF Ecore, for example domain-specific languages as defined in [9] or even Ecore instance models themselves.

Figure 2 shows an example UML class diagram used for modeling a small part of a *vehicle rental company*. The company owns several vehicles of different types (cars and trucks) that can be rented by a customer. Vehicles and customers each own specific attributes. They are identified within the company using distinct id numbers. All attributes have simple types (`String` and `Integer`) except for attribute *Vehicle::renter* whose type is class *Customer*.

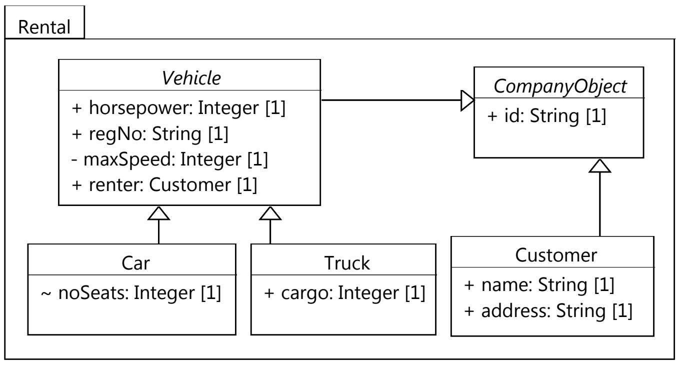


Figure 2: Example UML class model *VehicleRentalCompany*

In our example, we consider project-specific modeling conventions which recommend to express a class attribute that is typed by another class as an association end. If this convention is violated, i.e. there is a corresponding model smell that hints to the absence of quality aspect *conformity*, UML refactoring *Change attribute to association end* can be used to remove this smell. The refactoring consists of a number of preconditions that have to be checked before the refactoring can take place. For example, it has to be checked whether the attribute has another class as type. Due to space limitations, we do not explain the entire specification of this refactoring in detail.

In *EMF Refactor*, model refactorings can be specified using the new EMF model transformation tool Henshin [9]. Henshin uses pattern-based rules which can be structured into nested transformation units with well-defined operational semantics. *EMF Refactor* uses Henshin's model transformation engine for executing the refactoring as well as Henshin's pattern matching algorithm to detect violated preconditions.

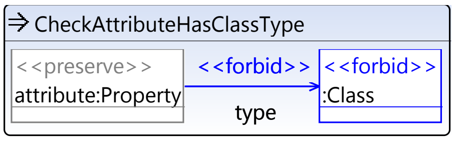


Figure 3: Henshin rule for checking precondition *Attribute has Class type*

Figure 3 shows the rule defining the mentioned precondition violation specified on the abstract syntax of UML. The pattern defines a *Property* node named *attribute* that represents the contextual element of the refactoring, i.e. the attribute that has to be changed to an association end. Furthermore, the pattern specifies a UML *Class* as attribute type. The class as well as the *type* reference are tagged by ⟪*forbid*⟫, i.e. this pattern must **not** be found in the model. So, the rule specifies the violated precondition from above.

The rule for executing refactoring *Change attribute to association end* is shown in Figure 4. Again, *Property* node *attribute* represents the contextual element of the refactoring. It becomes a member end of a newly created *Association* whose name is given by parameter `associationname`. Furthermore, another *Property* is created representing the second end of the new association. The name of this association end is given by parameter `rolename` and its type is set to the owning class of the contextual attribute.

EMF Refactor provides a wizard-based refactoring specification process. The process is triggered from within the context menu of an arbitrary model element whose type represents the contextual element of the refactoring and provides all necessary information about the specific meta model. Then, the refactoring designer has to give a name to the refactoring specified. Furthermore, a plug-in project has to be chosen for code generation purposes. Afterwards, the designer has to select the specification mode. Since the application module of *EMF Refactor* uses the Eclipse Language Toolkit (LTK) technology [10], a refactoring requires up to three parts, either implemented in Java or using model transformation specifications as presented above. The parts of a refactoring specification reflect a primary application check for a selected refactoring without input parameters, a second one with parameters and the proper refactoring execution. Finally, *EMF Refactor* generates Java code specific to the refactoring and extends the list of supported refactorings using the extension point technology of Eclipse.

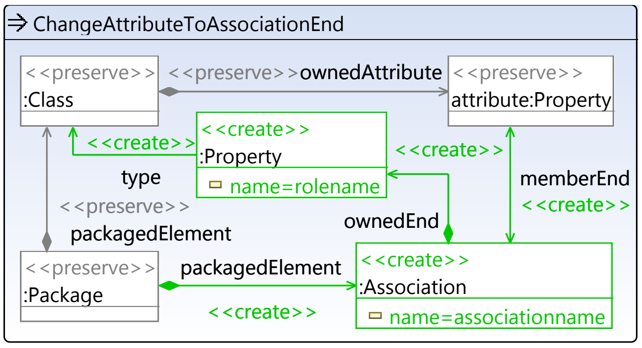
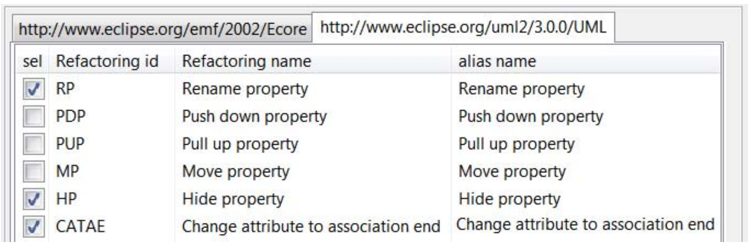


Figure 4: Henshin rule for executing UML model refactoring *Change Attribute to Association End*

3. EMF Model Refactoring Application

EMF Refactor supports project specific refactoring configurations to bundle those refactorings whose application may be most appropriate for the specific modeling purpose, for example analysis models in early development phases.

Figure 5 shows an example configuration of a refactoring group named *Conformance to modeling conventions*. The registered refactorings are listed wrt. a corresponding meta model and can be selected to join the specific refactoring group. Already configured refactoring groups can be selected by an arbitrary Eclipse project to customize project-specific refactoring needs.


http://www.eclipse.org/emf/2002/Ecore | http://www.eclipse.org/uml2/3.0.0/UML

sel	Refactoring id	Refactoring name	alias name
✓	RP	Rename property	Rename property
☐	PDP	Push down property	Push down property
☐	PUP	Pull up property	Pull up property
☐	MP	Move property	Move property
✓	HP	Hide property	Hide property
✓	CATAE	Change attribute to association end	Change attribute to association end

Figure 5: Existing refactorings can be configured into project-specific refactoring groups.

EMF Refactor provides refactoring invocation within the standard tree-based EMF instance editor, graphical editors generated by GMF (Graphical Modeling Framework) [11], and textual model editors based on Xtext [12]. The application of a model refactoring mirrors the three-fold specification of refactorings based on LTK as described in the previous section.

After specifying a trigger model element such as attribute *Vehicle::renter* in Figure 2, refactoring specific initial conditions are checked. Then, the user has to set all parameters, for example the names of the new association and association end in our example refactoring *Change attribute to association end*. Due to space limitations, we do not show the parameter input dialog here. Then, *EMF Refactor* checks whether the user input does not violate further conditions. In case of erroneous parameters a detailed error message is shown. If the final check has passed, *EMF Refactor* provides a preview of the changes using EMF Compare.

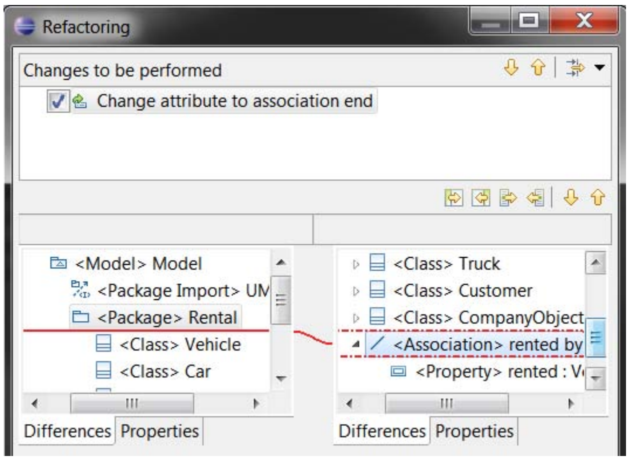


Figure 6: Result preview of UML refactoring *Change Attribute to Association End*

Figure 6 shows the resulting EMF Compare dialog using a tree-based model view. The left hand side shows the original model whereas the right hand side presents the refactored model. Model changes are highlighted by colored connections. Here, the right hand side shows the newly created association named *rented by* including the newly created association end named *rented*. Since a modeler is typically not aware of the tree-based representation of the model, it is up to future work to have a preview in the visual notation in the case that the refactoring has been triggered from within a graphical editor. Last but not least, these changes can be committed and the refactoring can take place. Figure 7 shows the UML class model *VehicleRentalCompany* after application of UML refactoring *Change Attribute to Association End* on attribute *Vehicle::renter* that is now presented as an end of the new association *rented by*.

4. Related work

There is a variety of literature on quality assurance techniques for software models available, often lifted from corresponding techniques for code. Most of the model quality assurance techniques have been developed for UML models. Classical techniques such as software metrics, code smells and code refactorings have been lifted to models, especially to class models. Model refactorings have been developed in e.g. [4], [13] and [14].

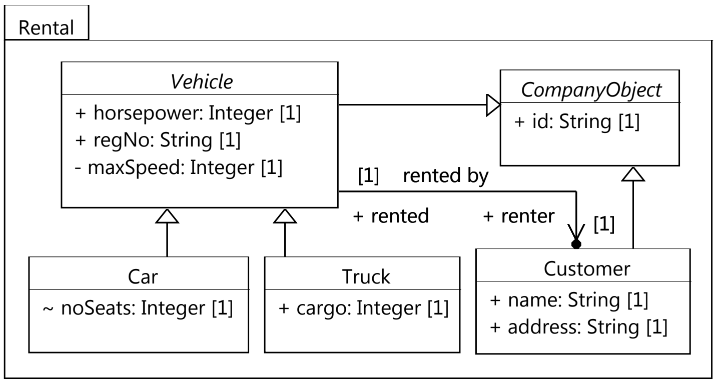


Figure 7: Example UML class model *VehicleRentalCompany* after application of UML refactoring *Change Attribute to Association End* on attribute *Vehicle::renter*

Considering UML model refactoring, there is nearly no tool support available yet. However, some research prototypes for model refactoring are discussed in the literature. Most of them are no longer maintained.

In [5] for example, Porres describes the execution of UML model refactorings as sequence of transformation rules and guarded actions. He presents an execution algorithm for these transformation rules and constructed an experimental, meta model driven refactoring tool, that uses SMW, a scripting language based on Python, for specifying the UML model refactorings.

Since EMF has evolved to a well-known and widely used modeling technology, it is worthwhile to provide model quality assurance tools for this technology. There are further tools for the specification of EMF model refactorings available as e.g. the Epsilon Wizard Language [15]. We compare our approach with other specification tools for EMF model refactorings in [16]. In contrast to EWL, we provide a specification frame for refactorings which allows different specification mechanisms. Especially, we propose Henshin, a model transformation approach based on graph transformation concepts which supports more correctness checks than EWL. In contrast to EWL, we further use the LTK technology for homogeneous refactoring execution in Eclipse including a result preview, for example. A new approach for EMF model refactoring is presented in [17]. The authors propose the definition of EMF based refactoring in a generic way, however do not consider the comprehensive specification of preconditions. Our experiences in refactoring specification show that mostly preconditions cannot be defined generically (see [9] for a more complex refactoring with elaborated precondition checks).

5. Conclusion

In this paper we presented *EMF Refactor*, a new Eclipse incubation project providing specification and application of model refactorings based on the Eclipse Modeling Framework. Actually, model refactorings are defined by the model transformation language Henshin or by Java code specifications. It is planned to support further specification alternatives for refactoring designers being not familiar with Henshin but other techniques like EWL. Currently we are implementing a comprehensive refactoring suite for UML2EMF and Ecore models. Future work also includes conceptual work on the design of a graphical language definition for composite refactorings including editor support, code generation, and refactoring application. Composite refactorings are meant to be build up from already specified refactorings. Furthermore, we are currently implementing a framework for automated tests of model refactorings.

EMF Refactor covers the automation of only one step in a model quality assurance process consisting of further quality assurance techniques like model metrics and model smells supported by tools *EMF Metrics* and *EMF Smell*. It is up to future work to combine these tools with *EMF Refactor*, especially to support refactorings as quick-fixes for model smells, in order to provide an integrated tool environment for syntactical model quality assurance of EMF based models.

[1] F. Fieber, M. Huhn, B. Rumpe, Modellqualität als Indikator für Softwarequalität: eine Taxonomie, Informatik Spektrum 31 (5) (2008) 408–424.

[2] M. Genero, M. Piattini, C. Calero, A Survey of Metrics for UML Class Diagrams, Journal of Object Technology 4 (9) (2005) 59 – 92.

[3] C. F. Lange, Assessing and Improving the Quality of Modeling: A series of Empirical Studies about the UML, Ph.D. thesis, Department of Mathematics and Computing Science, Technical University Eindhoven (2007).

[4] G. Sunye, D. Pollet, Y. Le Traon, J. Jezequel, Refactoring UML models, in: Proc. UML 2001, Vol. 2185 of LNCS, Springer-Verlag, 2001, pp. 134–148.

[5] I. Porres, Model Refactorings as Rule-Based Update Transformations, in: G. B. P. Stevens, J. Whittle (Ed.), Proc. UML 2003: 6th Intern. Conference on the Unified Modeling Language, LNCS, Springer, 2003, pp. 159–174.

[6] T. Arendt, S. Kranz, F. Mantz, N. Regnat, G. Taentzer, Towards Syntactical Model Quality Assurance in Industrial Software Development: Process Definition and Tool Support, in: Software Engineering 2011, SE 2011, Karlsruhe, Germany, LNI, GI Bonn, 2011, to appear.

[7] EMF, Eclipse Modeling Framework, http://www.eclipse.org/emf.

[8] EMF Refactor, http://www.eclipse.org/modeling/emft/refactor.

[9] T. Arendt, E. Biermann, S. Jurack, C. Krause, G. Taentzer, Henshin: Advanced Concepts and tools for In-Place EMF Model Transformation, in: Model Driven Engineering Languages and Systems, 13th International Conference, MoDELS 2010. Proceedings, LNCS, Springer, 2010, pp. 121–135.

[10] The Language Toolkit, http://www.eclipse.org/articles/Article-LTK.

[11] GMF, Graphical Modeling Framework, http://www.eclipse.org/modeling/gmf.

[12] Xtext, http://www.eclipse.org/Xtext/.

[13] A. Pretschner, W. Prenninger, Computing refactorings of state machines, Software and Systems Modeling 6 (4) (2007) 381–399.

[14] S. Markovic, T. Baar, Refactoring OCL Annotated UML Class Diagrams, Software and Systems Modeling 7 (2008) 25–47.

[15] D. S. Kolovos, R. F. Paige, F. Polack, L. M. Rose, Update transformations in the small with the epsilon wizard language, Journal of Object Technology 6 (9) (2007) 53–69.

[16] T. Arendt, F. Mantz, L. Schneider, G. Taentzer, Model Refactoring in Eclipse by LTK, EWL, and EMF Refactor: A Case Study, http://www.modse.fr/modsemccm09/doku.php?id=Proceedings (2009).

[17] J. Reimann, M. Seifert, U. Amann, Role-Based Generic Model Refactoring, in: Model Driven Engineering Languages and Systems, LNCS, Springer, 2010, pp. 78–92.