Defining and Checking Model Smells: A Quality Assurance Task for Models based on the Eclipse Modeling Framework[☆]

Thorsten Arendt[a], Matthias Burhenne[a], Gabriele Taentzer[a]

[a]*Philipps-Universität Marburg, FB12 - Mathematics and Computer Science, Hans-Meerwein-Strasse, D-35032 Marburg, Germany*

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 Smell*, a prototype Eclipse plug-in providing specification and detection of smells in models based on the Eclipse Modeling Framework.

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

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 model 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. Model smells occur in model parts that are potential candidates for improvements, i.e. they are not synonyms for problems but are worthy of an inspection. Here, the project-specific

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

Email addresses: arendt@mathematik.uni-marburg.de (Thorsten Arendt), matthias.burhenne@gmx.de (Matthias Burhenne), taentzer@mathematik.uni-marburg.de (Gabriele Taentzer)

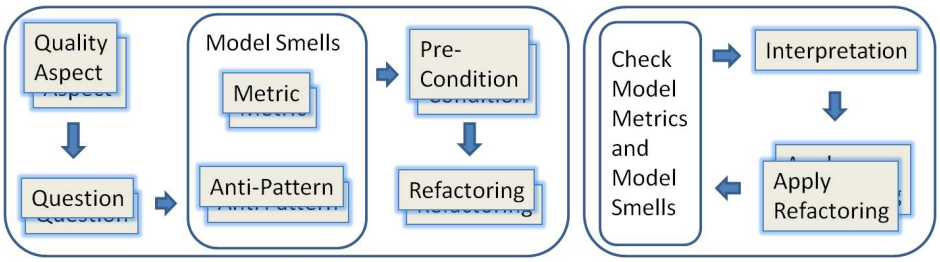

Figure 1: Project-Specific Quality Assurance Process - Specification (left side) and Application (right side)

process can reuse general metrics, smells, and refatorings, as well as special ones specific for the intended modeling purpose. After formulating quality checks, the specified 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 **Eclipse plug-in *EMF Smell* supporting specification and detection of smells wrt. specific EMF based models**.

This paper is organized as follows: In the next section, the model smell specification process using *EMF Smell* is presented. In Section 3, we describe how model smells are detected and reported in *EMF Smell*. Finally, related work is discussed and a conclusion is given.

2. Specification of EMF Model Smells

As already mentioned, *EMF Smell* supports specification and detection of smells wrt. EMF based models. In the following, we use an example UML2EMF model to demonstrate the capabilities of *EMF Smell*. Please note, that *EMF Smell* 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.

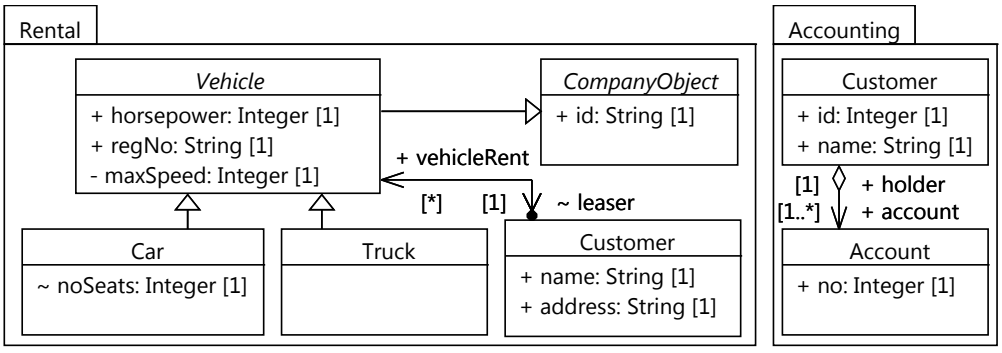

Figure 2: Example UML class model *VehicleRentalCompany*

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 (see package *Rental*). Furthermore, vehicles and customers each own specific attributes. They are identified within the company

using distinct id numbers. For accounting issues, a customer has at least one account. The last statement is modeled separately in package *Accounting*.

Considering model quality aspect *consistency* wrt. the modeling language used and wrt. modeling guidelines to be used, *Multiple definitions of classes with equal names* [3], *No specification* [8], *Concrete superclass* [3], *Class without attribute*, and *Attribute name overridden* are suitable model smells for quality checking. These smells are detectable by the existence of specific anti-patterns in the abstract model syntax. Other smells can be detected by metric benchmarks. For example, if we consider a project-specific modeling convention that recommends to hide attributes as often as possible, we can use UML model metric *Attribute hiding factor* (AHF) [2]. This metric is defined as ratio between the number of non-public attributes and the total number of attributes. In the following, we consider pattern-based model smells that are currently supported by *EMF Smell*. Here, the definition of (anti-) patterns is based on the corresponding meta model.

In *EMF Smell*, pattern-based model smells can be specified using the new EMF model transformation tool *Henshin* [9] that uses pattern-based rules that can be structured into nested transformation units with well-defined operational semantics. *EMF Smell* uses Henshin's pattern matching algorithm to detect rule matches that can be found in the model. The matches found represent the existence of the corresponding model smells.

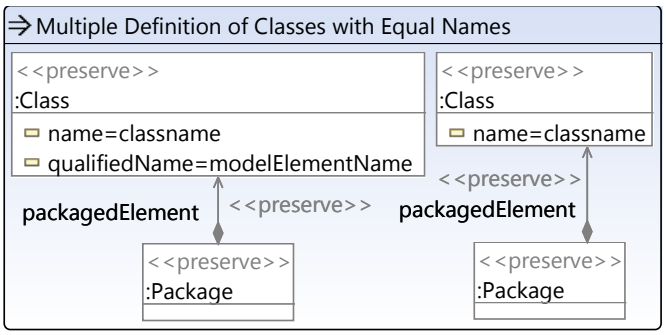

Figure 3: Henshin pattern rule specifying UML model smell *Multiple Definition of Classes with Equal Names*

Figure 3 shows the pattern rule for model smell *Multiple definitions of classes with equal names* specified on the abstract syntax of UML. The pattern defines two classes owned by distinct packages. Furthermore, the rule uses variable `classname` for specifying the equality of the names of the two involved classes. The qualified name of one of these classes is bound to rule variable `modelElementName` which is used for smell reporting purposes (see next section).

The pattern rule for model smell *No direct specification* is shown in Figure 4. This pattern specifies two classes with direct inheritance relationship. The superclass is abstract whereas the subclass is not. However, the subclass is tagged by ⟪*forbid*⟫, e.g. this pattern has **not** to be found in the model. So, the rule specifies an abstract class which does not have direct concrete subclasses. Again, the qualified name of abstract superclass is bound to rule variable `modelElementName` which is used for smell reporting purposes as described in the next section.

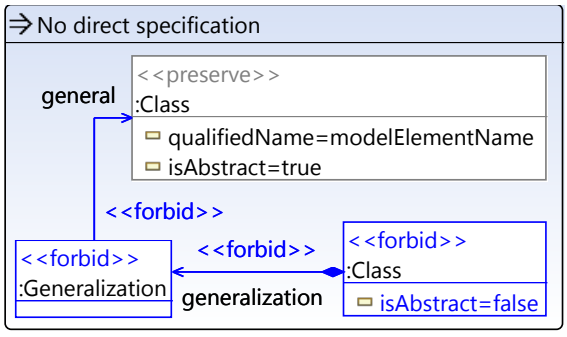

Figure 4: Henshin pattern rule specifying UML model smell *No Specification*

EMF Smell provides a wizard-based smell specification process triggered by the main workbench menu. First, the smell designer has to type in smell specific parameters *id*, *name*, and *description*. Furthermore, the corresponding meta model has to be selected as well as an existing plug-in project for code generation purposes. Afterwards, the designer has to select the specification mode. Either a Henshin file defining a pattern rule as shown in

Figures 3 and 4 is selected, or the generation of Java code skeletons without concrete smell specification implementation is chosen. Finally, *EMF Smell* generates Java code specific to the specified smell and extends the list of supported model smells using the extension point technology of Eclipse.

3. EMF Model Smell Detection and Reporting

EMF Smell supports project specific smell configurations to select those model smells which are appropriate to analyze specific modeling purposes. Here, the registered EMF model smells are listed wrt. the corresponding meta model as shown in Figure 5.

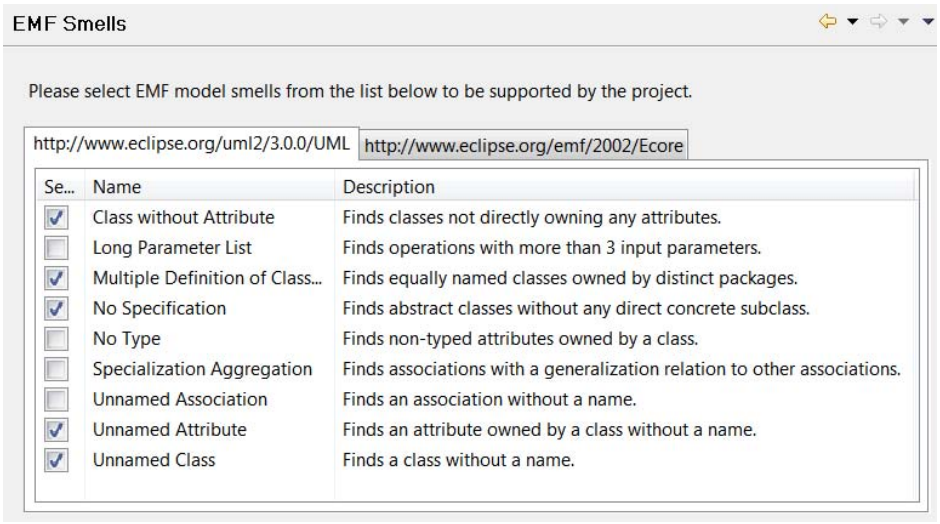

Figure 5: Project configuration for EMF model smells

The checking process is triggered from within the context menu of the model file to be analyzed. *EMF Smell* checks the existence of each configured model smell wrt. the corresponding meta model and presents the detected smells in a special result view. In this view, each model smell found contains a time stamp to trace detected smells over time. Furthermore, *EMF Smell* provides an XML export of its analysis results.

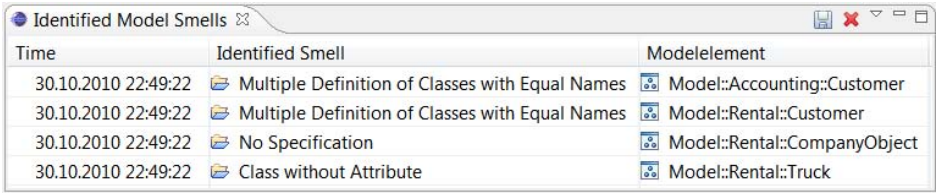

Figure 6: EMF Smell result view after detecting four smells on UML class model *VehicleRentalCompany* (see Figure 2)

Figure 6 shows the corresponding result view after analyzing the example model in Figure 2. Altogether four specified model smells were detected. Now, it is up to the model reviewers to interpret the results wrt. model changes, e.g. by the application of suitable model refactorings.

4. Related work

Code smells have been developed the last 20 years. In [10] for example, Fowler presents his standard work on refactoring and code smells. He describes smells for object-oriented systems extensively and suggests refactorings to

get rid of them. There are several tools available which support code smell detection. For example, *Checkstyle* [11] automates the process of Java checking code wrt. specific coding standards. 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 smells have been developed in e.g. [8] and [3]. To the best of our knowledge, related tools for smell detection in EMF based models are not yet available.

5. Conclusion

In this paper we presented *EMF Smell*, a prototype Eclipse plug-in providing specification and calculation of smells wrt. specific models based on the Eclipse Modeling Framework. Actually, model smells are defined using model transformation language Henshin or by Java code specifications. It is planned to support further smell specification alternatives for smell designers being not familiar with Henshin but other techniques like OCL (Object Constraint Language). Current work also consists of implementing a comprehensive smell suite for UML2EMF and Ecore models. Furthermore, it is planned to provide advanced smell reporting like highlighting of model elements involved in a detected model smell.

EMF Smell covers the automation of only one step in a model quality assurance process consisting of further quality assurance techniques like model metrics and model refactorings. Considering these techniques further tool support is available, namely *EMF Metrics* and *EMF Refactor* [12]. It is up to future work to combine these tools with *EMF Smell* (supporting metric-based smells or quick-fix refactorings, for example) 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] A. J. Riel, Object-Oriented Design Heuristics, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[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] M. Fowler, Refactoring - Improving the Design of Existing Code, Addison-Wesley, Reading/Massachusetts, 1999.

[11] Checkstyle, http://checkstyle.sourceforge.net/index.html.

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