EMF Metrics: Specification and Calculation of Model Metrics within the Eclipse Modeling Framework[☆]

Thorsten Arendt[a], Pawel Stepien[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 Metrics*, a prototype Eclipse plug-in providing specification and calculation of metrics wrt. models based on the Eclipse Modeling Framework.

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

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 that 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. Metrics are mathematical models used for measuring. In software engineering, metrics are utilized for measuring quality aspects of different artifacts (in our case software models). Here, the project-specific process can reuse general metrics, smells, and refatorings, as well as special ones

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

Email addresses: arendt@mathematik.uni-marburg.de (Thorsten Arendt), pawelstep@googlemail.com (Pawel Stepien), taentzer@mathematik.uni-marburg.de (Gabriele Taentzer)

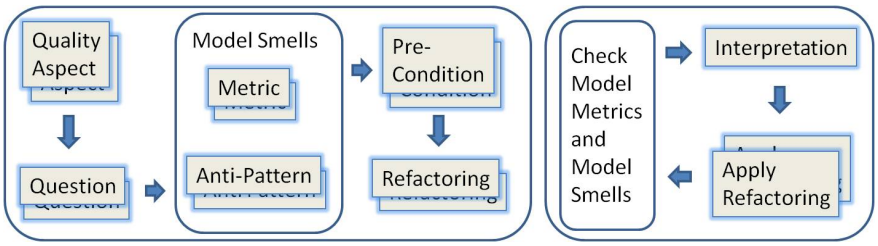

Figure 1: Project-Specific Quality Assurance Process - Specification (left side) and Application (right side)

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 Metrics* supporting specification and calculation of metrics wrt. specific EMF based models**.

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

2. Defining EMF Model Metrics

As already mentioned, *EMF Metrics* supports specification and calculation of metrics wrt. EMF based models. In the following, we use an example UML2EMF model to demonstrate the capabilities of *EMF Metrics*. Please note, that *EMF Metrics* can be used on arbitrary models whose meta models are instances of EMF Ecore, for example domain-specific languages as defined in [8] or even Ecore instance models themselves.

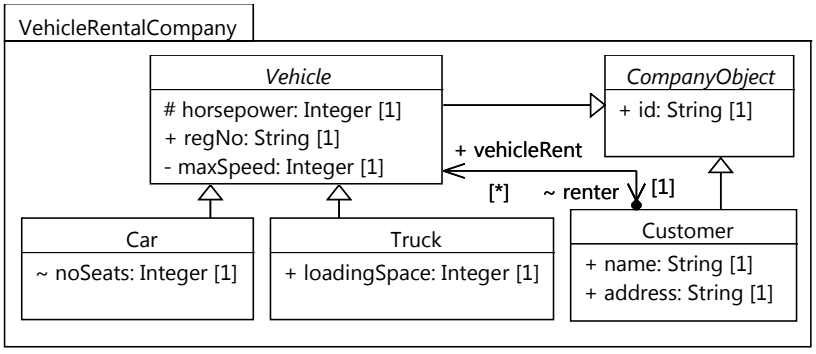

Figure 2: Example UML class model *VehicleRentalCompany*

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 several customers. Vehicles and customers each own specific attributes. They are identified within the company using distinct id numbers.

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 our case, we adapt this metric on UML classes in order to identify those ones that violate quality aspect *conformity* since the AHF value differs from 1. A class with an AHF value different to 1 does not satisfy the afore mentioned modeling convention since its number of non-public attributes does not correspond to its total number of attributes, i.e. the class owns public (non-hidden) attributes.

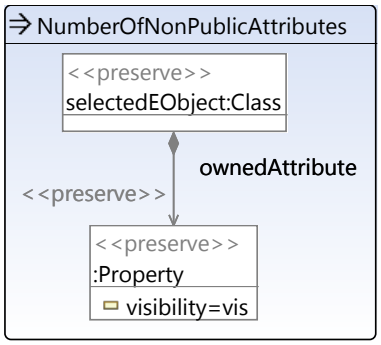

Figure 3: Henshin pattern rule specifying UML metric *Number of non-public attributes*

To define UML model metric AHF, it is helpful to specify metrics *Number of non-public attributes* (NONPA) and *Number of attributes* (NOA). Our approach for metrics definition is to define a pattern first and to count its occurrences in the model thereafter. In *EMF Metrics*, metrics NONPA and NOA can be specified using the new EMF model transformation tool *Henshin* [8] that uses pattern-based rules that can be structured into nested transformation units with well-defined operational semantics. *EMF Metrics* uses Henshin's matching algorithm to determine the total number of pattern matches that can be found in the model representing the value of the corresponding model metric.

Figure 3 shows the pattern for metric NONPA specified on the abstract syntax of UML. The pattern defines a node named *selectedEObject* of type *Class* that represents the contextual model element for calculating the corresponding metric. Furthermore, the rule specifies the pattern of an attribute that is owned by this class. According to the UML meta model, this attribute is presented by meta model type *Property*. Its visibility is bound to variable `vis` in order to evaluate the given value by attribute condition `vis!="public"` specifying non-public visibility. Given the contextual UML class by *selectedEObject*, the total number of matches of this pattern provides the value of metric NONPA.

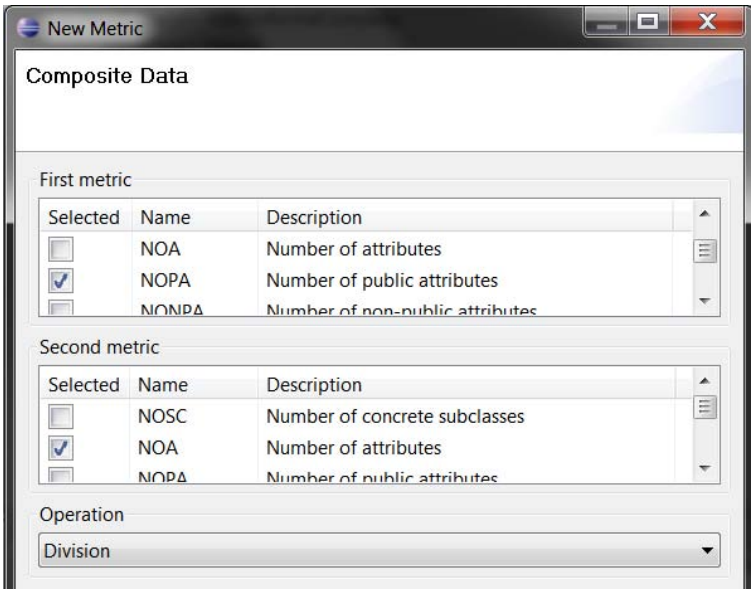

Figure 4: Compositional specification of UML metric *Attribute hiding factor*

For defining compositional metrics, *EMF Metrics* supports a combination of existing ones. Here, the metric designer simply has to select the involved existing metrics as well as the appropriate arithmetic operation. Currently, *EMF Metrics* provides the binary arithmetic operations *sum*, *subtraction*, *multiplication*, and *division*. For specifying e.g. UML metric *Attribute hiding factor*, metrics NONPA and NOA are combined using the binary arithmetic operation *division* (see Figure 4). Of course, only those existing metrics are presented whose contextual elements correspond to the contextual element of the new compositional metric.

EMF Metrics provides a wizard-based metrics specification process. The process is triggered from within the context menu of an arbitrary model element whose type represents the contextual element of the metric. Then, the metric designer has to type in metric specific parameters *id*, *name* and *description*. Furthermore, a plug-in project has to be chosen for code generation purposes. Afterwards, the designer has to select the specification mode. Either a Henshin file defining a pattern rule as shown in Figure 3 has to be selected, or a combination of existing metrics is specified as presented in Figure 4. Finally, *EMF Metrics* generates Java code specific to the specified metric and extends the list of supported model metrics using the extension point technology of Eclipse.

3. Calculating EMF Model Metrics

EMF Metrics provides project specific metrics configurations to select those metrics whose calculations may be most appropriate for the specific modeling purpose. The registered model metrics are listed wrt. the corresponding meta model and contextual element as shown in Figure 5.

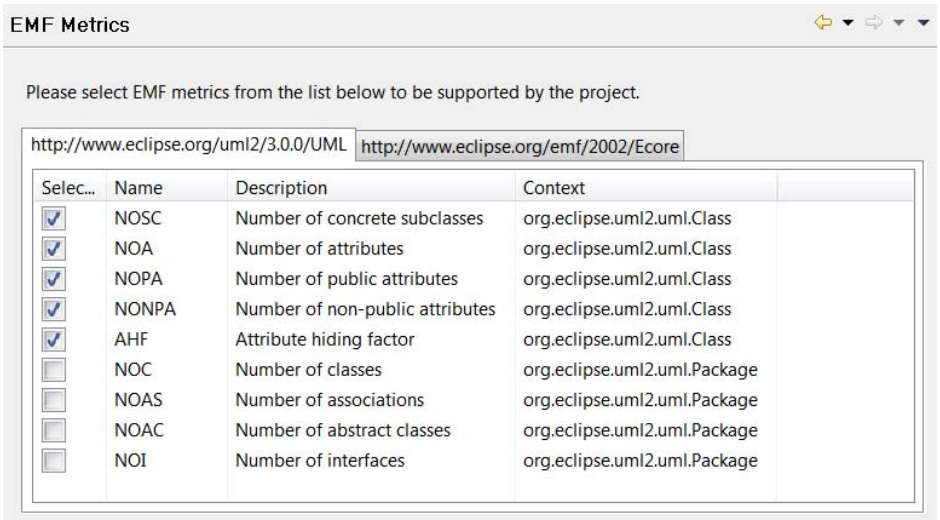

Figure 5: Project configuration for EMF model metrics

The calculation of a configured metrics set is triggered from within the context menu of a specific model element being the contextual element. *EMF Metrics* calculates the value of each selected model metric and presents the values in a special result view. Figure 6 shows the corresponding result view after calculating five different metrics on abstract class *Vehicle* (see model in Figure 2). This class owns altogether four attributes *horsepower*, *regNo*, *maxSpeed*, and *renter*. Attribute *regNo* is public whereas the other attributes are hidden. So, the AHF value of class *Vehicle* is evaluated to 0.75. Please note, that we only considered direct (non-inherited) attributes in the metrics specification presented in section 2.

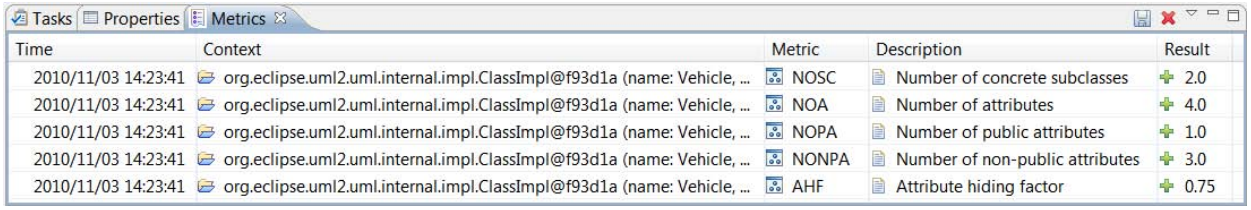

Figure 6: EMF Metrics result view after calculating five different metrics on Class *Vehicle*

Currently, each result is marked with '+' indicating a 'good' value. It is up to future work to support project-specific customizing for each metric in order evaluate results concerning 'good', 'moderate' and 'bad' values. In the result view, each result contains a time stamp to trace metric values over time. Furthermore, *EMF Metrics* provides an XML export of its results.

4. Related work

Considering software metrics a lot of work was done during the last 30 years for very different purposes. For example, in [9] Tom DeMarco presents an effective strategy for measuring software development costs. His credo is: *"You can't control what you can't measure."* A variety of tools exist for measuring metrics for program code. An overview on existing metrics tools can be found at [10]. There is already quite some 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 metrics have been developed in e.g. [2] and [11]. A representative for tools supporting the calculation of metrics wrt. UML models is *SDMetrics* [12]. Mostly, these tools are not integrated in a specific UML design tool, i.e. the corresponding UML model has to be exported respectively imported using its XMI representation. To the best of our knowledge, related tools for metrics calculation on EMF based models are not yet available.

5. Conclusion

In this paper we presented *EMF Metrics*, a prototype Eclipse plug-in providing specification and calculation of metrics wrt. specific models based on the Eclipse Modeling Framework. Actually, metrics are specified by using model transformation language Henshin for pattern definition and their matching or by combining existing metrics using an arithmetic operation. It is planned to support further specification alternatives for metric designers being not familiar with Henshin but other techniques like OCL (Object Constraint Language) or even Java. Current work also consists of implementing a comprehensive metrics suite for UML2EMF and Ecore models. Furthermore, it has to be evaluated whether additional combination operations should be supported.

EMF Metrics covers the automation of only one step in a model quality assurance process consisting of further quality assurance techniques like model smells and model refactorings. Considering these techniques further tool support is available, namely *EMF Smell* and *EMF Refactor* [13]. It is up to future work to combine these tools with *EMF Metrics* 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] 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.

[9] T. DeMarco, Controlling Software Projects: Management, Measurement, and Estimates, Prentice Hall, 1986.

[10] Niwot Ridge Resources, http://www.niwotridge.com/Resources/PM-SWEResources/MetricsTools.htm.

[11] H. van Elsuwe, D. Schmedding, Metriken fuer UML-Modelle, Informatik Forschung und Entwicklung 18 (1) (2003) 22–31.

[12] SDMetrics, http://www.sdmetrics.com/.

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