Measuring the Explicitness of Modeling Concepts in Metamodels *

Horst Kargl
Business Informatics Group,
Vienna University of Technology
Favoritenstrasse 9-11
1040, Vienna, Austria
kargl@big.tuwien.ac.at

Michael Strommer
Business Informatics Group,
Vienna University of Technology
Favoritenstrasse 9-11
1040, Vienna, Austria
strommer@big.tuwien.ac.at

Manuel Wimmer
Business Informatics Group,
Vienna University of Technology
Favoritenstrasse 9-11
1040, Vienna, Austria
wimmer@big.tuwien.ac.at

ABSTRACT

Metamodels represent the abstract syntax of modeling languages. However, they do not explicitly define which modeling concepts the user can use in the notation of the language. Thus, simply counting the number of classes in the metamodel is not appropriate for altering the number of modeling concepts available to the user. This paper addresses this issue with the definition of a metric for describing the ratio between the number of concepts explicitly represented by classes in the abstract syntax and the concepts available in the notation. In particular, this metric is important when bridging two modeling languages, because the hidden concepts in the abstract syntax have to be made explicit in the transformation rules.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Measurement, Language

Keywords

metamodel metrics, modeling languages

1. INTRODUCTION

Evaluation of models is a hard and ambiguous task. Whether a model can be asserted as semantically correct within a certain domain of discourse always depends on the viewpoint of the observer. In order to gather objective facts about models, the use of metrics is necessary. Metrics have a long history in software development as a quality measure. Measured value can be interpreted for itself, or it can be combined to create aggregated metrics for stating a more abstract conclusions. During the triumphal procession of Object Oriented Programming (OOP) in the last one and a half decades and the subsequent development of Object Oriented Modeling (OOM) techniques, a lot of effort was made to develop metrics for object-oriented models (OO-metrics). OO-metrics allow to make statements about the quality of software models [8][14][7].

With the advent of Model Driven Engineering (MDE) [12], the need of formal metamodels for modeling languages arises and consequently the need for metrics for metamodels. Basically a metamodel can be seen a s a model; it also consists of classes, relations, inheritance, etc. However, the intension of a metamodel (the abstract syntax) is different from that of most M1 models. A metamodel is an object oriented language definition. Therefore the instances of a metamodel are again models with a graphical notation, the concrete syntax. Nevertheless, metamodels can be treated as models for a specific domain, the domain of modeling language definition. Hence, most of the metrics for OOM can be applied to metamodels [6].

In this paper we introduce a new metric, which discovers the *explicitness* of a metamodel. We define explicitness in the context of metamodels as the number of concepts in the modeling language that are first class concepts in the metamodel. This definition is based on the assertion that the number of first class concepts in the abstract syntax can differ from that in the concrete syntax. For example consider the modeling concept *Attribute* in the UML class diagram. In the UML 2.0 class diagram metamodel there exists no first class definition for *Attribute* . It is hidden in the class *Property* [5]. But in the concrete syntax of UML 2.0 class diagrams, there exists a notation for the concept *Attribute*.

The contribution of this paper consists of a new metric for measuring the explicitness of a metamodel with the help of the notation elements available in the concrete syntax. Therefore we count concrete classes in the metamodel and divide them with the number of notation elements in the concrete syntax. This ratio specifies the need of refactoring the metamodel to get the implicit concepts explicit [5].

*This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-810806.0

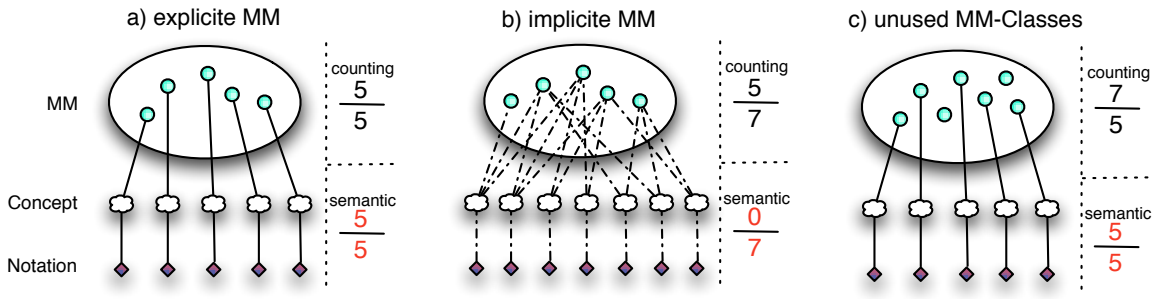

Figure 1: Three characteristics of the balance value

The remainder of this paper is structured as follows. Section 2 gives an introduction of our approach with a description of the calculation, its interpretation, some side effects, and a solution to reduce these side effects. Section 3 describes the execution of the metric in the context of the UML 1.4 and the UML 2.1 metamodel. Section 4 discusses the related work. Finally, Section 5 concludes with an outlook on future work.

2. CALCULATING THE EXPLICITNESS OF METAMODELS

Modeling languages have specific modeling concepts, which can be expressed with its notation elements (the concrete syntax). The composition of modeling concepts is defined in the metamodel (the abstract syntax). These modeling concepts, which are used in the concrete syntax, do not necessarily have an exact counter part in the abstract syntax. Concepts in the metamodel are often reused with the help of attributes and associations to other concepts. We call this phenomenon concept hiding [5]. Our aim is to get a metric to estimate the explicitness of metamodels. The Explicitness of MetaModel (EM^2) metric can be calculated by counting all concrete classes of a metamodel and dividing it by the number of notation elements on the concrete syntax.

$$EM^2(as, n) = \frac{count(as.concreteClasses())}{count(n.elements())}$$

A metamodel does not only consist of concrete classes but of abstract classes as well. Since abstract classes have no conceptual representation on the concrete syntax, they cannot be instantiated. For this reason abstract classes are not counted.

2.1 Interpretation of the Metric EMM

In general, we can make three assertions about the ratio between the number of concrete classes in the metamodel and the number of notation elements.

EM² value	**Interpretation**
< 1	Concept deficit
$= 1$	Concept equilibrium
> 1	Concept redundancy | overload

A graphical representation is given in figure 1. The ellipses represent the metamodels with their first class concepts. The clouds represent the semantic concepts of the modeling language. The diamonds represent the notation elements. The links between these three elements depict the reference between the metamodel and its notation. The top right fraction (cf. *counting* in figure 1) represents the ratio by counting MM-concepts and notation elements without including further information. The bottom right fraction (cf. *semantic* in figure 1) represents the semantically correct ratio that describes the ratio of concepts and their representation on the concrete syntax. To determine the bottom right fraction, further information is necessary, that is included in the links between metamodel and notation. In the following we discuss the three aforementioned distinct cases of the EM^2 metric.

- *Concept deficit*: An EM^2 value smaller than one means that there exist more concepts in the concrete syntax than in the abstract one (see figure 1b). The reason for this is that the concepts are hidden in the metamodel. In the field of model transformations, which are defined on the metamodel, it is easier to map one metamodel to another if no hidden concepts exist in the metamodel. A balance smaller than one is an indicator for the need of refactoring [5].
- *Concept equilibrium*: An EM^2 value of one means that there are exactly as many concepts in the metamodel as in the concrete syntax (see figure 1a).
- *Concept overload*: An EM^2 value greater than one can have two reasons. The first one is a notation in the concrete syntax with more than one concept in the metamodel. Two concepts in the metamodel with similar semantics is a non-realistic assumption and can be left out. The second reason are metamodel concepts, which do not represent a notation element (see figure 1c).

2.2 Side Effects of the Metric

Automatically applying the metric to a modeling language may entail some side effects. Only concrete classes of the metamodel and all notation elements that form the concrete syntax are counted. Depending on the metamodel, a concrete class must not necessarily have a notation element.

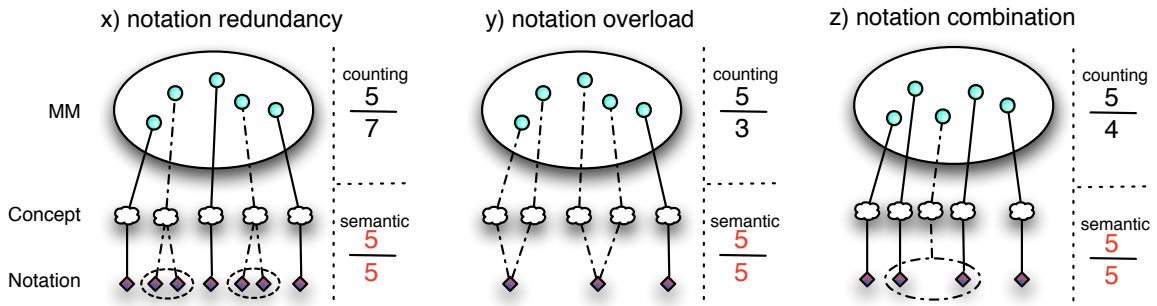

Figure 2: Side effects of the metric

This increases the denominator of the fraction and results in a higher EM^2 (the upper right fraction in figure 1c). This would be interpreted as a more explicit metamodel. The bottom right fraction in figure 1c) depict the semantic ratio between concepts, metamodel classes and its notation elements.

Counting metamodel classes with no representation in the concrete syntax can be prevented by following the mapping between a metamodel concept and its notation. If there is a link with no constraint, the metamodel class has a concrete notation element. Otherwise, the class hides some information and is not a first class element for this concept. A constraint link is a mapping between a metamodel class and a notation element having further restrictions, e.g., that a association to another metamodel class ought to be set, or that an attribute of the metamodel class must have a specific value. With this further information, most of the side effects and the resulting misinterpretation of the balance can be avoided.

On the level of the concrete syntax, it is possible that a language concept has two or more notation elements, like the interface notation in UML. By counting all these notation elements, the numerator of the fraction increases and the balance becomes lower, see figure 2x. This would lead to interpret the metamodel as less explicit, with more hidden concepts. Evaluating the mappings between the notation and the metamodel, as aforementioned, can avoid this circumstance. If there exists a link without a constraint between a metamodel class and two notation elements this two notation elements are counted as one.

One and the same notation elements can be used for different concepts (2y). This result in a lower numerator of the fraction and leads to the interpretation of an overspecified metamodel. Again following the links between notation elements and metamodel elements is a solution to this problem.

Notation elements represent a concept of the modeling language, but it is possible that the combination of two or more notation elements stands for a different concept which has no explicit notation. This changes the balance to a higher ratio if the concept is explicit in the metamodel but could not be counted on the concrete syntax because combined notation elements are not countable (2z). In this case following the links between metamodel concepts and notation elements provides no solution. An approach to cope with this problem is to analyze one or more concrete examples (M1 models) and analyze how the notations can be combined.

3. ANALYSING UML 1.4 AND UML 2.0

The UML metamodel 1.4.2 [10] and 2.1 [11] for class modeling were chosen as test cases for the EM^2 metric. Strictly counting concrete classes in the metamodel and predefined notation elements from the specifications, we found the following estimates for the EM^2 value.

Metamodel version	$\mathbf{EM^2}$ value
UML 1.4.2	$\approx 0,77$
UML 2.1	$\approx 0,65$

These estimations have however to be taken with care because of possible inaccuracies in counting notation elements. There exist several notation elements where it is not clear whether to count them as one or each separately, as we did in our evaluation scenario. As an example for this kind of problem you might consider the *Dependency* relationship and its various stereotyped notation elements attached to it. Also we have not eliminated arising side effects, except for the *notation combination* side effect.

3.1 Discussion the Metric Results

Although the results are not totally unbiased they seduce that both metamodels rely on implicit concepts. Furthermore we can say there is a greater implicitness incorporated into the UML 2.1 metamodel than in the UML 1.4.2 metamodel. The class *Attribute* represents just one concept in UML 1.4.2 that has been made implicit in UML 2.1. In the following we will now go more into details and illustrate the characteristics and side effects of our metric.

As a potential source of implicitness of concepts we discovered the usage of enumerations, which are heavily applied in both metamodels. The definition of the EM^2 metric does not involve the count of enumerations and their literals. But these literals often represent a notation element, often in

combination with some classes. For example, the enumeration *AggregationKind* is used in both metamodels to distinguish between three different notation elements, that is to say regular association, aggregation, and composition.

The depicted side effect *notation overload* in figure 2, appeared in the UML 2.1 metamodel in the case of the meta classes *ElementImport* and *PackageImport*. Both classes make use of the same notation element. The meaning between the two can only be made unambiguous when considering the connected model elements, i.e., *Classes* or *Packages*.

Notation redundancy could also be recognized in both metamodels. Consider the interface class as an example, which can be graphically represented by two different means. Another example would be the various possible notation forms of an association (with a diamond in the middle or without, or with an arrow or without). The algorithm described above would filter such redundancies to eliminate the problem and concentrate on concepts instead of graphical representations.

We also encountered the problem of unused concrete meta classes that do not define a general notation. The responsibility is instead delegated to some other classes, that can be subclasses of the class under consideration. For example, the class *Parameter* in UML 2.1 has no direct link to any notation element. The class *Operation* therefore defines the notation for its parameters. Similar to the count of unused meta classes is the count of general notation elements that have no concrete class in the metamodel. The class *MultiplicityElement*, that is declared abstract, specifies a general notation for multiplicities, which can be further specialized in corresponding subclasses. The EM^2 metric takes the notation into account, but omits the abstract class, leading to a rare side effect, that we call *standalone notation*.

Combining notation elements is common practice in the UML metamodels. Take as an example the notation for a stereotype, that is composed of the notation of a simple class and the name of the stereotype within guillemets. When computing our metric for the two metamodels we counted each combined notation element as individual to avoid the side effect resulting from notation combination.

4. RELATED WORK

Best to our knowledge, there has been no work on metrics for explicitness of metamodels and our work is the first study on this topic. However, our work is mainly influenced by two orthogonal research directions, on the one hand by metrics for UML class diagrams and on the other hand by metrics for Ontologies.

Metrics for UML class diagrams are mostly based on metrics for OO programs. This is due to the close connection of UML class diagrams to OO programming languages like Java. In [14] six different metrics for UML class diagrams are analyzed and compared whereas the question arises which model elements, e.g., classes, attributes, and associations, have impact on the complexity of a class diagram. In [9] the metrics are more generically defined based on graph structures. Again, the metrics operate on quantitative characteristics, e.g., node count, edge count, and path length, and then these single metrics are combined to higher-order metrics.

Summarizing these proposed metrics for UML class diagrams mostly focus on the quantitative analysis of model elements, thus the metrics only measure the explicit definitions. Our work is different, because we look for implicit concepts which are hidden in combinations of model elements. In addition, we are analyzing language definitions and therefore we study the relation between modeling concepts, abstract syntax, and concrete syntax, which is certainly not applicable to UML class diagram models.

In [15] and [3] various metrics for ontologies are discussed which mainly measure the structural dimension in the same way as with OO models reflecting the fact that most ontologies are also represented in an object-oriented manner. Additionally to the structural measurement in Gangemi et al. [2] measurements for the functional dimension and usability, as well as a NLP-driven evaluation are introduced. Furthermore, the OntoClean approach [13] tries to detect both formal and semantic inconsistencies in an ontology. This perception goes along with our that counting the number of elements of certain types is not sufficient to specify the complexity of a model.

Our work is different to the proposed ontology metrics in that with our metric we are able to indicate how many concepts are implicitly represented which is due to the exploitation of the abstract to concrete syntax mapping which is metamodeling depending and not an ontology topic. Nevertheless, many ontology techniques are promising for the semantic evaluation of models and metamodels which is subject to future work.

The most related work is [4] in which OO metrics are applied to assess five versions of UML metamodels. The authors propose metrics for the stability of UML metamodels and for the design quality of UML metamodels such as reusability, flexibility, and understandability, which are computed from single measures.

Our work is different due to two facts. First, we also incorporate the notation of the modeling language, and second, we analyze which modeling concepts are missing in the abstract syntax as first class definitions. However, it is interesting that in [4] the computed value for understandability of UML 2 is much worse compared to its predecessor. Furthermore, it would be very interesting to compute the measurements for the design quality before and after applying our proposed refactoring patterns as introduced in [5].

5. CONCLUSION AND FUTURE WORK

In this paper we introduced the EM^2 metric for determining the explicitness of metamodels. The EM^2 metric helps to analyze a metamodel and to get an idea about how many concepts are hidden in it. With this metric it is possible to detect which metamodel classes are potential hiding points for concepts by analyzing the mapping between the notation elements and the metamodel classes. One drawback of EM^2 is that an implementation of the modeling language with its notation elements is necessary to eliminate the aforemen-

tioned side effects. Otherwise, the metric possibly delivers biased ratios from the modeling language.However, for tool support the implementing of modeling language is necessary and therefore no additional costs arise.

The motivation for this metric is the emerging field of Model Driven Engineering (MDE) and model transformation, which are defined on the metamodel layer. The EM^2 metric is useful for determining the degree to which metamodels can be matched automatically according to their explicitness. Facilitating automatic matching of two metamodels, which have a less explicitness, is a nearly impossible task. Future work is a prototypical implementation of the metric within the Eclipse Graphical Modeling Framework (GMF [1]). The hidden concepts, once detected using EM^2, can then be made explicit using the refactoring approach described in [5]. The implementation of the EM^2 metric is further motivated through the hard and error prone task of extracting notation elements of the UML specification, that is due to the informal and fragmented definition of the mapping between the notation and the abstract syntax.

6. REFERENCES

[1] Graphical Modeling Framework GMF, June 2006.

[2] M. C. J. L. Aldo Gangemi, C. CAtenacci. A theoretical framework for ontology evaluation and validation. In *Semantic Web Applications and Perspectives Semantic Web Applications and Perspectives (SWAP) - 2nd Italian Semantic Web Workshop*, Trento, Italy, 2005.

[3] A. Cross, V.; Pal. Metrics for ontologies. In *Fuzzy Information Processing Society, 2005. NAFIPS 2005. Annual Meeting of the North American*, pages 448–453, June 2005.

[4] Y. Jiang, W. Shao, L. Zhang, Z. Ma, X. Meng, and H. Ma. On the classification of uml's meta model extension mechanism. In *UML*, pages 54–68, 2004.

[5] G. Kappel, E. Kapsammer, H. Kargl, , G. K. Thomas Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer. Lifting metamodels to ontologies - a step to the semantic integration of modeling languages. ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy, 2006.

[6] H. Ma, W. Shao, L. Zhang, Z. Ma, and Y. Jiang. Applying oo metrics to assess uml meta-models. volume 3273, pages 12–26, 2004.

[7] E. Manso, M. Genero, and M. Piattini. No-redundant metrics for uml class diagram structural complexity. In M. M. Johann Eder, editor, *Advanced Information Systems Engineering, 15th International Conference, CAiSE 2003, Klagenfurt, Austria, ISBN 3-540-40442-2*, volume 2681 of *Lecture Notes in Computer Science*, pages 127–142. Springer, jun 2003.

[8] C. C. Marcela Genero, Mario Piattini. A survey of metrics for uml class diagrams. *Journal of Object Technology*, 4(9):59–92, November-December 2005.

[9] T. Mens and M. Lanza. A graph-based metamodel for object-oriented software metrics. *Electr. Notes Theor. Comput. Sci.*, 72(2), 2002.

[10] OMG. *Unified Modeling Language Specification Version 1.4.2, formal/04-07-02*. OMG, 2004.

[11] OMG. *Unified Modeling Language: Superstructure version 2.1 ptc/2006-04-02*. OMG, April 2006.

[12] D. C. Schmidt. Model driven engineering. *IEEE Computer Society*, February 2006.

[13] C. Welty, R. Kalra, and J. Chu-Carroll. Evaluating ontological analysis. In *In Proceedings of the ISWC-03 Workshop on Semantic Integration*, 2003.

[14] T. Yi, F. Wu, and C. Gan. A comparison of metrics for uml class diagrams. *SIGSOFT Softw. Eng. Notes*, 29(5):1–6, 2004.

[15] H. Yoa, A. M. Orem, and L. Etzkorn. Cohesion metrics for ontology design and application. *Journal of Computer Science*, 1(1):107–113, 2005.