What do Metamodels Really Look Like?

James R. Williams, Athanasios Zolotas, Nicholas Matragkas,
Louis M. Rose, Dimitios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack

Department of Computer Science, University of York, UK
{james.r.williams,amz502,firstname.lastname}@york.ac.uk

Abstract. Model-Driven Engineering promotes the use of tailor-made modelling languages for software and systems engineering problems, with *metamodels* that encapsulate domain knowledge. Despite the importance of metamodelling in MDE, there is little empirical analysis of metamodels. What are the common characteristics of metamodels? Do modellers follow best practices? How do metamodels evolve over time? How does the size and structure of a metamodel affect the models that conform to it? This paper takes a first step towards answering these questions by automatically analysing the structural characteristics of a corpus of more than 500 publicly available Ecore metamodels.

1 Introduction

A common activity in Model-Driven Engineering (MDE) is *metamodelling* – the process of capturing the concepts and structures of a particular domain in a *metamodel* in order to construct models of that domain. Metamodels exist for general modelling languages (GMLs), such as UML, and for a range of domain-specific modelling languages (DSMLs), created to address specific software engineering domains. However, there is little guidance on the desirable or undesirable characteristics of metamodels for GMLs or DSMLs. There has been much research into the quality of models, but there is little empirical analysis of metamodels. If we can analyse different properties and calculate various metrics of metamodels, we may to be able to identify and detect good and bad practice, and understand the ways in which people are commonly structuring their metamodels today. The work here was motivated by the need to understand the common structural aspects of metamodels in order to tailor a model generation tool towards generating realistic metamodels for testing purposes.

In this paper we reveal common characteristics of metamodels that we have identified from an automated analysis of a corpus of over 500 publicly-available Ecore [9] metamodels. This is a first step: once we can analyse metamodels in different contexts and for different purposes, we can identify patterns of metamodelling best practice, and metamodel refactorings that facilitate model operations such as transformation. We can also develop an understanding of how metamodels evolve over time, and seek to control the complexity of evolving metamodels to minimise the effects on model artefacts, operations and tools. Our plan is to produce a set of standard metrics and analyses for metamodels

– similar to what exists in other domains (e.g. OO source code metrics) – and develop a supporting automated metamodel measurement workbench.

We use a general-purpose model management language, the *Epsilon Object Language* [6] (EOL), to compute counts or descriptive statistics on metamodel characteristics. The analysis shows the range of structural characteristics of metamodels, identifies some of the common practices of metamodellers – the most used parts of the metamodelling language, and the ways in which domain concepts are typically expressed – and raises many further questions about the commonalities and differences across the metamodelling corpus.

Section 2 introduces a set of metrics, focusing for now on structural analysis of metamodels. Section 3 presents the results of analysing the corpus of metamodels, and explores how the structure of one metamodel changed during its evolution. Section 4 describes related research.

2 Foundations: Structural Properties of Metamodels

The metrics considered in this paper focus on structural properties of metamodels – understanding how people structure their metamodels and answering the question *what do metamodels really look like?* The 19 metrics are examples of what can be achieved using simple EOL programs. We use EOL as it provides an executable query language, akin to OCL, that can easily be executed on metamodels. Our metrics are grouped into two categories: those related to meta-classes, and those concerning meta-features (attributes and references). The full list of metrics can be found on our website: `www.jamesrobertwilliams.co.uk/mm-analysis`. We summarise them now.

Our initial set of meta-class metrics focuses on the frequencies of meta-classes with various properties in a metamodel. This includes the *total number of meta-classes* metric, which gives an indication of the *size* of a metamodel, whilst the *total number of concrete meta-classes* and the *total number of abstract meta-classes* metrics provide more detail. Incidentally, though unintentionally, our metrics overlap and extend the metrics defined in recent work by Ma et al [7]. We also define metrics to inspect the number of features in a meta-class. Featureless classes may be considered to be bad design; detecting these in metamodels would highlight bad practice. We define metrics on two kinds of featureless meta-class: *immediately* featureless classes – those that have no attributes or references, but may inherit features from a superclass; and *completely* featureless classes have absolutely no features. Further metrics might explore the frequency of reference features, as compared to attribute features, or the distribution of features across hierarchies. In addition to counting, we can create descriptive statistics such as means and medians. We also calculate the average number of features per class, broken down by feature kind (attribute or reference). These metrics can be used to analyse whether there is a tendency to create many small classes, develop 'God' classes, or distribute features across classes.

The metrics concerning meta-features are *global* – referring to the number of occurrences of features in an entire metamodel and illustrate how metamodellers commonly define the data (attributes) in metamodels and how they relate meta-classes to one another. These metrics include: counts of the total number of features in a metamodel (attribute, references, and combined); the types of references being defined (*containment* or *non-containment* – *compositions* or *associations* in UML terms); and examinations of the upper multiplicity bounds of references.

3 Analysis: What do Metamodels Really Look Like?

This section uses the metrics overviewed in the previous section to analyse, firstly, a large number of metamodels in an attempt to the common structural properties of metamodels. By computing these properties, we hope to inform the community of how people are modelling domains and attempt to learn how to improve current practice. Secondly, we analyse the evolution of a large metamodel over 11 minor versions and see how these properties change over time. The analysis script, the corpus of metamodels, and more detailed results are available online at: `www.jamesrobertwilliams.co.uk/mm-analysis`.

3.1 Analysing the Corpus of Metamodels

We have accumulated a corpus of 537 publicly available Ecore [9] metamodels. The corpus is made up of metamodels collected from GitHub, Google Code, the AtlantEcore Zoo, the EMFText Zoo, and from internal projects[1]. The corpus includes many well known modelling languages – such as the UML, DODAF, and Marte – as well as metamodels for many programming languages such as Java, C#, C, and Pascal, and many domain-specific metamodels. We then collated the scores and now describe the results. Due to space limitations, graphical visualisations of these statistics can be found at the web page above.

Meta-class Metrics The median total number of meta-classes in the corpus is 13, with a mean of 39.3, a maximum of 912, and a minimum of one. This suggests that metamodels (at least, in this corpus) are often fairly small. Twelve of the 537 metamodels have a single meta-class. Five of these metamodels are meaningless and should be removed, four were extensions of other metamodels, and three were domain-specific metamodels which also defined custom data types or enumeration types. Although small, a single-class metamodel can still define a suitable modelling language for some domains. The corpus showed that abstract meta-classes were not popular: 44% of metamodels did not contain a single meta-class denoted as being abstract. Furthermore, 96% of the corpus has fewer than 20 abstract meta-classes, whereas only 69% of the corpus has fewer than 20 concrete meta-classes. This is arguably due to the small average size of the corpus:

[1] GitHub: `github.com`; Google Code: `code.google.com`; AtlantEcore Zoo: `www.emn.fr/z-info/atlanmod/index.php/Ecore`; EMFText Zoo: `www.emftext.org/index.php/EMFText_Concrete_Syntax_Zoo`

smaller metamodels are likely to contain only concrete classes, whereas large metamodels are more likely to utilise abstract classes. Featureless classes were uncommon: 58% of the corpus has no completely featureless classes, and 27% have no immediately featureless classes. Interestingly, in the UML metamodel (developed by the Eclipse UML2 project (`http://www.eclipse.org/uml2/`)) 50 of the 227 meta-classes were immediately featureless, 40 of those were concrete. Immediately featureless classes are much more common than completely featureless ones, and it is more likely that these immediately featureless classes are concrete. Further analysis would likely show that these are specialisations of abstract classes, perhaps to provide some extra semantics to the hierarchy.

Meta-feature Metrics The median number of meta-features per metamodel is 23.5, with a mean of 69.2, a maximum of 2410, and a minimum of zero. Metamodels in the corpus commonly have more references (median 13.5, mean 43.0) than attributes (median 8, mean 26.2). The average metaclass has 2.1 features: 1.15 references and 0.95 attributes. The large number of featureless classes present in the corpus affects these data. If we exclude featureless classes when calculating the average features per class, we obtain the same distributions, although the mean number of features per meta-class increases slightly to 2.3, with 1.3 references and 1.0 attributes.

On average metamodels contain more non-containment references (median 6, mean 27.3) than containment references (median 5, mean 15.7). With respect to reference upper bounds, we find that they are set to 'one' 52% of the time, to 'many' 47% of the time, and are explicitly given a value just 1% of the time. The trend towards selecting 'many' as opposed to explicitly defining an upper bound might be attributed to the inherent uncertainty in modelling [11] (of course, sometimes specifying an upper bound as 'many' is perfectly acceptable and not related to domain uncertainty).

3.2 Analysing the Evolution of a Metamodel

The previous analysis considered only one fixed state of each metamodel, and doesn't capture how these properties change over time. Understanding how metamodels evolve can provide many insights, such as highlighting smells or anti-patterns [4]. Moreover, developers of metamodelling tools can use the information to provide the most appropriate support for practitioners, such as for the development of model migration [8] tools. We analyse 11 versions of the *Graph* metamodel, part of the *Graphical Modeling Framework* [5], an Eclipse project for developing graphical editors for modelling languages. We analyse versions 1.23 to 1.33 inclusive. More detailed results can be found on our web page.

The analysis exposed some major structural refactorings that occurred at version 1.29. The total number of meta-classes stays constant, however the number of concrete classes decreases by 25%. These structural refactorings also manifest in the total number of features, increasing at version 1.29, whilst the average number of features per classes stays fairly constant. Perhaps most revealing, however, are the featureless classes metrics. Many newly introduced classes were immediately featureless, and the change in numbers of abstract and concrete meta-classes suggests that concrete-classes were refactored to abstract. The number of totally featureless classes, however, stayed constant, suggesting that meta-classes were introduced as specialisations, and these refactorings were a reorganisation of the class hierarchy.

This analysis only considers the evolution of a single metamodel. It would be interesting to discover whether the behaviour shown in this example is commonly found in other metamodels, or to see whether we can discover patterns of evolution by analysing a large number of metamodel evolutions. Unfortunately, however, these intermediate models may not be available in all cases.

3.3 Threats to Validity

Although we have analysed a large number of metamodels, we need to be careful with our conclusions. Almost all of the metamodels in the corpus were publicly available and downloaded from the internet. These metamodels may not be representative of the metamodels that are used in industry. Many of the metamodels were of well known modelling languages (e.g. UML, CORBA's IDL, DODAF) which are used in industry, but the corpus may not represent the bespoke modelling languages that are developed in practice. Furthermore, all of the metamodels that were analysed were Ecore metamodels. Different metamodelling technologies may have differing properties, and the metamodelling language itself may cause the kinds of behaviour shown in this paper. Ecore, however, is arguably the current de facto modelling language and is equivalent to EMOF so the insights that it offers are still useful.

4 Related Work

While there is a significant amount of work in the field of analysis of MDE artefacts, the majority of the related work we have encountered has a different focus to this paper. The closest work to ours is Cadavid et al [2] who present an empirical analysis of the ways MOF and OCL are used together. They define metrics to analyse the complexity of 33 metamodels, their constraints, and the coupling between the two. The work in this paper aims to complement the work of Cadavid et al. with deeper analysis of the metamodel structure (as opposed to its relationship with its constraints). Vepa et al. [10] measure a set of metamodels that is stored in the Generative Modeling Technology/ATLAS MegaModel Management (GMT/AM3) Repository. This work focuses on the model repository and the measuring technique, rather than the presentation of the results of the analysis. Finally, Arendt et al. [1] describe an Eclipse plugin, *EMF Metrics*, that can be used to assess the quality of EMF metamodels based on nine quantitative criteria. The aforementioned approaches focus mainly on model quality (as with software), while we (and Cadavid) are interested in understanding the usage of metamodelling languages and how metamodels are constructed.

5 Conclusion

In this paper we have posited the need for a deeper understanding of metamodels. We illustrate structural analysis on a corpus of over 500 metamodels, gaining insight into how metamodels are commonly structured, and how they evolve over time. We are now in a position to start the analysis of good and bad practice in metamodelling, for GMLs or DSMLs, and in different model management contexts. To facilitate development of further metrics, we are creating a metrics metamodel. We plan to create a web-based automated metamodel measurement workbench that allows users to upload and analyse their own metamodels, which will automatically augment to the results given here. We plan to devise a comprehensive set of metrics, and develop state-of-the-art analyses for metamodels, taking inspiration from similar domains, such as bad smell detection [4] and design patterns [3].

Acknowledgements

This research was part supported by the EPSRC, through the Large-Scale Complex IT Systems project (EP/F001096/1) and by the EU, through the Automated Measurement and Analysis of Open Source Software (OSSMETER) FP7 STREP project (318736).

References

1. T. Arendt, P. Stepien, and G. Taentzer. Emf metrics: Specification and calculation of model metrics within the eclipse modeling framework. In *BENEVOL 2010*, 2010.
2. J. Cadavid, B. Baudry, and B. Combemale. Empirical evaluation of the conjunct use of MOF and OCL. In *EESSMod 2011*, 2011.
3. H. Cho and J. Gray. Design patterns for metamodels. In *Proc. DSM'11*, 2011.
4. M. Fowler. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, Boston, MA, USA, 1999.
5. R. C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit.* Addison-Wesley Professional, 2009.
6. D. S. Kolovos, R. F. Paige, and F. A. C. Polack. The Epsilon Object Language (EOL). In *ECMDA-FA*, volume 4066 of *LNCS*, pages 128–142. Springer, 2006.
7. Zhiyi Ma, Xiao He, and Chao Liu. Assessing the quality of metamodels. *Frontiers of Computer Science*, pages 1–13, 2013.
8. L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. C. Polack. An analysis of approaches to model migration. In *Proc. MoDSE-MCCM 2009*, October 2009.
9. D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF Eclipse Modeling Framework.* The Eclipse Series. Addison-Wesley, second edition, 2009.
10. E. Vépa, J. Bézivin, H. Brunelière, and F. Jouault. Measuring model repositories. In *Proc. MoDELS/UML 2006, Workshop on Model Size Metrics*, October 2006.
11. J. R. Williams, F. R. Burton, R. F. Paige, and F. A. C. Polack. Sensitivity analysis in model-driven engineering. In *Proc. MODELS 2012*, 2012.