Towards the use of slicing techniques for an efficient invariant checking

Wuliang Sun, Benoit Combemale, Robert B. France

Towards the Use of Slicing Techniques for an Efficient Invariant Checking

Wuliang Sun,Benoit Combemale, and Robert B. France

***Abstract*—In Model Driven Development (MDD), invariant checking involves determining whether a model is consistent with invariants defined in a metamodel. Such checking can improve developers' understanding of modeled aspects of complex systems and uncover structural errors in design models during the early stages of software development. General-purpose rigorous analysis tools that check invariants are likely to perform the analysis over the entire metamodel and model. Their scalability thus becomes an issue (e.g., the time used for checking can be up to several hours) with very large metamodels and models (e.g., more than 500,000 elements). In this paper we introduce model slicing within the invariant checking process, and use a slicing technique to reduce the size of checking inputs to improve the scalability of existing invariant checking tools. The evaluation we performed provides evidence that model slicing can significantly reduce the time to perform the invariant checking while preserving the checking results.**

***Index Terms*—UML, Model, OCL, Invariant Checking, Slicing**

I. Introduction

Model-driven development (MDD) is a paradigm that (1) promotes models as the major artifacts to drive the software development process, and (2) uses model transformation and code generation to bridge the gap between high-level design models and low-level implementations. In MDD, models are often used by code generators. Since design errors in the models may be propagated into the implementations via model-to-code transformation, it is very important to uncover design errors during the early stages of software development.

Invariant checking involves determining whether an instance (i.e., model) of a metamodel satisfies the well-formed rules (i.e., invariants) defined in the metamodel. At design time, tool-supported invariant checking (e.g., see the Eclipse OCL project [11]) provides instant feedback on models and can thus enhance the ability of developers to identify potentially costly problems in models before they are used to generate code. However, the scalability of existing techniques for invariant checking on large models becomes an issue. For example, as shown in experiments we conducted and described in this paper, checking invariants against instances (e.g., models with hundreds of thousands of elements) of a metamodel that includes 345 elements would take more than two hours. There is thus a need for techniques that support scalable invariant checking.

Slicing techniques [12] produce reduced forms of artifacts that can be used to support, for example, analysis of artifact properties. Slicing techniques have been proposed for different software artifacts, including programs (e.g., see [12]), and models (e.g., see [1] [5]). In the MDD area, model slicing techniques have been used to support a variety of modeling tasks, including model comprehension [1] , analysis [4][6][7], and verification [8][9].

In model slicing techniques, *slicing criteria* are input data used to determine the elements that are included in slices. Model slicing techniques typically proceed in two steps: (1) The dependency between model elements of interest (i.e., elements satisfying a *slicing criterion*) and the rest of a model is analyzed using heuristics related to a model's properties (e.g., the structure of a model); and (2) a fragment of the model consisting only of elements satisfying a slicing criterion, is extracted from the model.

In this paper we introduce the model slicing technique to the invariant checking process. The approach aims to improve the scalability of existing invariant checking tools.The approach is not intended to improve the existing invariant checking algorithms. Instead, the approach aims to reduce the size of the checking inputs to make the analysis more efficient. It means our approach preprocesses the input of the invariant checking process, and thus is agnostic to the checking technologies the software developers are working with. In this paper we focus on analyses that involve checking the consistency between a model and the invariants defined in a metamodel, and checking whether a model is a valid instance of a metamodel is out of scope of the paper. Thus the precondition of our approach is that the input model must be a valid instance of the input metamodel. This means that we assume the model conforms to the structural constraints (e.g., multiplicity constraints) defined in the metamodel, but may or may not satisfy the invariants defined in the metamodel.

II. Motivation

At design time, a typical invariant checking process goes as follows: (1) A software modeler designs a metamodel and defines a set of Well Formed Rules (WFRs) (i.e., invariants) ; (2) He creates a set of models that conform to the metamodel structure; (3) He checks models against the WFRs using invariant checking tools such as the Eclipse OCL checker [11]. The entire process works well for small models with hundreds of elements, and the modeler can receive the feedback from the checking tools within seconds or minutes.

However, given the growing complexity of software systems, models used to represent these complex systems will also grow significantly in size. While design models were built by hand in the early days of MDD, nowadays models with possibly more than one million elements can be built programmatically. For example, we used a reverse engineering

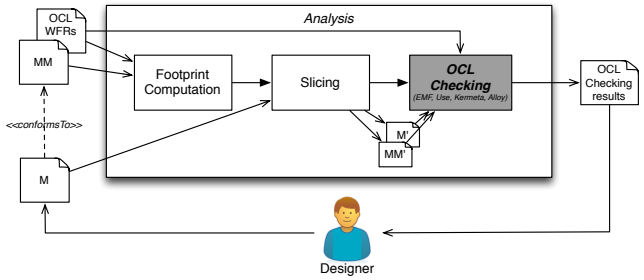

Fig. 1: Approach overview

tool, namely MoDisco [2], to generate Java models from multiple Eclipse platform plugins. These models could have up to one million elements. The checking time significantly goes up (e.g., more than two hours in the worse case scenario) for large models with hundreds of thousands of elements. This motivates the use of the scalable invariant checking approach in the context of large models.

Checking models against invariants does not need the entire metamodel and the full model to be present. Instead only a small part of the metamodel and model that are referenced by the invariants needs to be used for the invariant checking. In addition, a substantial number of invariants only reference part of the metamodel in which they are defined [3]. This motivates the use of the slicing technique in the context of invariant checking. The slicing technique thus can be used to reduce the size of the input metamodel and model to make the checking more efficient.

III. Approach

Figure 1 shows an overview of the proposed invariant checking approach. The input of the checking includes a metamodel (*MM*), a model (*M*), and one or many OCL invariants (*Well-Formed Rules*). First, the approach computes a footprint from the metamodel and OCL invariants. A footprint refers to part of a metamodel that contains all elements that affect the outcome of an operation [4]. In this paper a footprint refers to all metamodel elements that are directly referenced by the input OCL invariants. Second, the footprint serves as slicing criterion, and is used to generate a sliced metamodel (*MM'*) from the input metamodel. The sliced metamodel (*MM'*) includes (1) all the metamodel elements from the footprint, and (2) all the subclasses of the classes in the footprint. Third, the sliced metamodel (*MM'*) is used to generate a sliced model (*M'*) from the input model. The sliced model (*M'*) contains only model elements that are instances of metamodel elements in *MM'*. Finally, the sliced metamodel and model with the invariants are fed into the tools for invariant checking.

IV. Evaluation

We have developed a framework that provides: (1) an implementation of the model slicing technique; (2) an implementation for checking models against invariants defined in the metamodels. The framework was implemented using Java and the Eclipse Modeling Framework (EMF) [10]. Even though the evaluation framework builds upon Java and Eclipse, the slicing technique is not bound to a particular technological space, and it can be implemented using any language and framework.

We have evaluated our slicing technique to check whether (1) the slicing improves the efficiency of the invariant checking, and (2) the invariant checking results for the sliced models are the same as the unsliced models. The metamodel used for the evaluation is the Java metamodel. The models used for the evaluation were generated from 73 Eclipse plugins. The sizes of these 73 models range from 175926 to 993319 in terms of total number of object model elements including objects, links, and slots. Six invariants were used in the evaluation.

For each invariant, we measure the checking time used for unsliced metamodel and model (*CTUM*), the slicing time (*ST*), and the checking time used for sliced metamodel and model (*CTSM*). We use the Metric 1 below to evaluate the checking time speedup:

$$CheckingTimeSpeedup(CTS) = \frac{CTUM}{CTSM + ST} \quad (1)$$

The evaluation we performed provides evidence that the proposed slicing technique can significantly reduce the time to perform the invariant checking for the Java metamodel and its instances (e.g., achieve checking speedup ranging from 3 to 36) while preserving the checking results. Both the implementations of the slicing technique and the evaluation framework can be found in *https://github.com/sunwuliang/SlicingProject3.0*.

References

[1] A. Blouin, B. Combemale, B. Baudry, and O. Beaudoux. Kompren: modeling and generating model slicers. *Software & Systems Modeling*, pages 1–17, 2012.

[2] H. Brunelière, J. Cabot, G. Dupé, and F. Madiot. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, 56(8):1012 – 1032, 2014.

[3] J. Cadavid, B. Combemale, and B. Baudry. Ten years of Meta-Object Facility: an analysis of metamodeling practices. In *Technical Report by the Triskell Team at INRIA/IRISA*, pages 1–25, 2012.

[4] C. Jeanneret, M. Glinz, and B. Baudry. Estimating footprints of model operations. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 601–610. IEEE, 2011.

[5] H. Kagdi, J.I. Maletic, and A. Sutton. Context-free slicing of UML class models. In *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM)*, pages 635–638. Ieee, 2005.

[6] K. Lano and S. Kolahdouz-Rahimi. Slicing of UML models using model transformations. *In Model Driven Engineering Languages and Systems*, pages 228–242, 2010.

[7] K. Lano and S. Kolahdouz-Rahimi. Slicing techniques for UML models. *Journal of Object Technology*, 10, 2011.

[8] A. Shaikh, R. Clarisó, U.K. Wiil, and N. Memon. Verification-driven slicing of UML/OCL models. In *Proceedings of the IEEE/ACM international conference on Automated Software Engineering*, pages 185–194. ACM, 2010.

[9] A. Shaikh, U.K. Wiil, and N. Memon. Evaluation of tools and slicing techniques for efficient verification of UML/OCL class diagrams. *Advances in Software Engineering*, 2011, 2011.

[10] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2008.

[11] Eclipse OCL Project Team. Eclipse OCL Project. In *http://projects.eclipse.org/projects/modeling.mdt.ocl*. Eclipse Community, 2005.

[12] M. Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, pages 439–449. IEEE Press, 1981.