Formal Refactorings for Object Models

Rohit Gheyi
rg@cin.ufpe.br

Tiago Massoni
tlm@cin.ufpe.br

Informatics Center
Federal University of Pernambuco
Recife, Brazil

ABSTRACT

Both model and program refactorings are usually proposed in an *ad hoc* way because it is difficult to prove that they are sound with respect to a formal semantics. Even developers using refactoring tools have to rely on compilation and tests to guarantee the semantics preservation, which may not be satisfactory to critical software development. This work proposes a set of primitive structural semantics-preserving transformations for Alloy, a formal object-oriented modeling language. We use the Prototype Verification System (PVS), which contains a formal specification language and a theorem prover, to specify and prove the soundness of the transformations. Although our transformations are primitive, we can compose them in order to derive coarse grained transformations, such as a model refactoring to introduce a design pattern into a model. Moreover, proposing refactorings in this way can not only facilitate design, but also improve the quality of model refactoring tools.

Categories and Subject Descriptors

D.2.4 [**Software/Program Verification**]: Formal Methods; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Mechanical verification

General Terms

Design, Verification

Keywords

Model Refactoring, Object Models, Theorem Proving

1. INTRODUCTION

Evolution is an important and demanding software development activity, as the originally defined structure usually does not accommodate adaptations, demanding new ways to reorganize software. Modern development practices, such as program refactoring [3], improve programs while maintaining their original behavior, in order, for instance, to prepare software for change. Similarly, a *model refactoring* is a transformation that improves design structure preserving the semantics. They might bring similar benefits but with a greater impact on cost and productivity, since they are used in earlier stages of the software development process. For instance, they can be used to improve the analysis performance of modeling tools and introduce design patterns into a model, as shown elsewhere [4].

In current practice, even using refactoring tools, programmers have to rely on compilation and a good test suite [5] to ensure that it preserves the observable behavior [3]. In case of model refactorings, most proposed transformations rely on informal argumentation. Both program and model refactorings are usually proposed in an *ad hoc* way because it is difficult to prove that they are sound with respect to a formal semantics. So, simple mistakes may lead to incorrect transformations that might, for example, introduce inconsistencies to a model or make a program ill-typed. This may be unacceptable, especially for developing critical software systems. Numerous approaches prove refactorings based on an informal or formal argumentation. However, a few provide partial formal proofs; some refactorings are proved with respect to the type system [14], or based on a formal semantics stating that they preserve behavior [1].

Related work [2, 13, 9] has been carried out on semantics-preserving transformations for UML class diagrams. These approaches do not state in which conditions a transformation can be applied. Therefore, some transformations may not preserve semantics in some situations because some of them use a semi-formal UML semantics. Others partially define a semantics for UML but do not verify soundness of transformations, or do not consider OCL constraints. We conclude that it is important to prove soundness of transformations based on a formal semantics, in order to guarantee that a transformation preserves semantics. It is easy to make a small change in a model and make it inconsistent or introduce a type error.

2. GOAL STATEMENT

In this research, we propose a comprehensive set of primitive structural semantics-preserving transformations for Alloy [8]. Alloy is a formal object-oriented modeling language founded on relational logic that is similar to UML class diagrams annotated with OCL. Alloy models can be automatically analyzed, by means of the Alloy Analyzer tool.

We prefer to propose primitive localized fine-grained transformations that deals with one construct each time, since it is easier to prove that it preserves semantics. Although they are primitive, we can compose them and derive interesting coarse-grained transformations, such as model refactorings. Since these refactorings are derived using semantics-preserving transformations, they also preserve semantics. In this kind of work, it is important to prove that our set of transformations is relatively complete to derive a comprehensive set of model transformations. In order to show this property, we need to prove a reduction theorem, in the sense of allowing reduction of arbitrary Alloy specifications to a normal form. This normal form is expressed in a small subset of the language operators, following approaches adopted for object-oriented [1] and imperative languages [7], among others. In order to facilitate tool support, we aim at proposing transformations with syntactic enabling conditions.

3. APPROACH AND EVALUATION

Since we are proposing transformations that compare object models, first we need to propose an equivalence notion stating when two object models are equivalent. The common equivalence notion states that two object models are equivalent if they have the same semantics. This notion is useful, but not flexible enough to compare equivalent models with auxiliary elements [6]. Therefore, we need a more flexible equivalence notion. We intend to prove that each transformation not only preserves semantics, but also that it does not introduce any type errors or break the well-formedness properties of Alloy models. Consequently, we have to specify a formal semantics for Alloy.

PVS [11], which encompasses a formal specification language and a theorem prover, will be used to specify an equivalence notion and a formal semantics for Alloy, and to specify and prove our transformations [5]. A theorem prover will be used to increase our confidence, as manual proofs can be an error-prone activity. This aspect differentiates our approach, leading to visible benefits. Moreover, we intend to prove some properties about our equivalence notion. For example, we aim at not only proving that our equivalence notion is compositional, but also relating it to the traditional notion of Z refinement [15].

These transformations can be useful to refactor Alloy models with subtypes in order to improve the analysis performed by the Alloy Analyzer tool [4]. Moreover, they can be composed to derive coarse-grained transformations, such as model refactoring, which may be useful, for instance to introduce design patterns into a model. Since our transformations present syntactic enabling conditions, are sound and relatively complete, they can be useful to develop more reliable model refactoring tools. Additionally, the increasing interest in Model-Driven Architecture [12] has drawn attention to performing transformations on models, preferentially in an automatic fashion. We are especially interested in transformations regarding structural properties of domain models, which can be represented by Alloy models. In this context, these transformations can be mostly useful for carrying out PIM-to-PIM [12] semantics-preserving transformations. Other possible application lie on automatic synchronization between models and source code refactoring [10], development of improved tool support for refactoring, or even comparing whether the specification of two software components are equivalent from based on syntactic conditions [4].

So far, we have proposed an equivalence notion and some related properties, and specified a formal semantics and type system for Alloy in PVS [6]. Moreover, we have proposed and proved in PVS 14 model transformations, besides other 30 syntactic sugar transformations, and derived some coarse-grained transformations, such as optimizations [4]. As a future work, we intend to propose more transformations, prove the relative completeness of our transformations, derive more model refactorings, and prove some properties of our equivalence notion, such as compositionality.

Acknowledgments

We would like to thank Paulo Borba and the other members of Software Productivity Group, and all anonymous referees, whose appropriate comments helped improving the paper.

4. REFERENCES

[1] P. Borba et al. Algebraic Reasoning for Object-Oriented Programming. *Science of Computer Programming*, 52:53–100, 2004.

[2] A. Evans. Reasoning with UML class diagrams. In *2nd IEEE Workshop on Industrial Strength Formal Specification Techniques*, pages 102–113, 1998.

[3] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[4] R. Gheyi, T. Massoni, and P. Borba. Basic Laws of Object Modeling. In *3rd Specification and Verification of Component-Based Systems*, pages 18–25, USA, 2004.

[5] R. Gheyi, T. Massoni, and P. Borba. A Rigorous Approach for Proving Model Refactorings. In *20th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, USA, 2005.

[6] R. Gheyi, T. Massoni, and P. Borba. An Abstract Equivalence Notion for Object Models. *Electronic Notes in Theoretical Computer Science, Brazilian Symposium on Formal Methods*, 130:3–21, 2005.

[7] C. Hoare et al. Laws of programming. *Communications of the Association for Computing Machinery*, 30(8):672–686, 1987.

[8] D. Jackson et al. A micromodularity mechanism. In *Foundations of Software Engineering*, pages 62–73, Austria, 2001.

[9] K. Lano and J. Bicarregui. Semantics and transformations for UML models. In *UML'98 - Beyond the Notation*, pages 97–106, Mulhouse, France, 1998.

[10] T. Massoni, R. Gheyi, and P. Borba. A Model-driven Approach to Program Refactoring. Submitted for Application, 2005.

[11] S. Owre et al. PVS System Home Page. At http://pvs.csl.sri.com, 2005.

[12] R. Soley. Model Driven Architecture, November 2000. OMG Document 2000-11-05.

[13] G. Sunyé et al. Refactoring UML Models. In *4th UML Conference*, volume 2185 of *LNCS*, pages 134–148. Springer, 2001.

[14] F. Tip, A. Kiezun, and D. Baumer. Refactoring for Generalization Using Type Constraints. In *18th OOPSLA*, pages 13–26. ACM Press, 2003.

[15] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice Hall, 1996.