UML/OCL Verification In Practice

Jordi Cabot and Robert Clarisó

Universitat Oberta de Catalunya
{jcabot, rclariso}@uoc.edu

Abstract. In the MDD approaches, models become the primary artifact of the development process and the basis for code generation. Identifying defects early, at the model-level, can help to reduce development costs and improve software quality. There is an emerging need for verification techniques usable in practice, i.e. able to find and notify defects in real-life models without requiring a strong verification background or extensive model annotations. Some promising approaches revolve around the *satisfiability* property of a model, i.e. deciding whether it is possible to create a well-formed instantiation of the model. We will discuss existing solutions to this problem in the UML/OCL context. Our claim is that this problem has not yet been satisfactorily addressed.

1 Introduction

Model-driven development (MDD) advocates for the use of models as development artifacts. In this context, code is no longer written from scratch but synthesized from models (semi-)automatically. Therefore, any defect in the model will propagate into defects in the code. In MDD model correctness becomes a key factor in the quality of the final software product.

Although the problem of ensuring software quality has attracted much attention and research, it is still considered to be a Grand Challenge [1]. In this sense, this paper argues that this grand challenge must be adapted and extended to cover the verification of modeling notations commonly used in MDD approaches. In this field, it is essential to provide a set of tools and methods that helps in the detection of defects at the model-level and smoothly integrates in existing MDD-based tool-chains without an excessive overhead. Characteristics of existing tools, such as required designer interaction or manual model annotations seriously impair its usability in practice.

We will discuss existing approaches and their limitations to motivate that this problem is still unsolved and remains an important challenge in MDE. Throughout the paper, we will focus on the UML as an example of a MDD modeling language. However the contents of the paper also apply to all kinds of domain-specific modeling languages (DSMLs). Some of the problems identified herein already arise when just considering the graphical elements of the models. However, complexity increases when designers use textual languages (e.g. OCL) to improve the precision/formalization of the models. Note that all existing model-level verification approaches proceed by translating the models to a more formal language (e.g. Alloy). Therefore, characteristics (and limitations) of correctness techniques for UML/OCL models largely depend on the properties of available verification techniques for those underlying formal languages.

2 A definition of "correctness"

One of the first problems when discussing correctness at the model-level is the large number of existing correctness notions, according to many different criteria: static vs dynamic, inter-diagram vs intra-diagram, ...

A first degree of correctness can be that of *consistency* and *well-formedness*, checking that all uses of a model element (possibly in different diagrams) are consistent with its declaration and that the model as a whole can be expressed as a correct instantiation of its meta-model.

Even though this analysis provides an initial level of defect detection, it does not take into account the *semantic* correctness of the model being defined. By semantics, we consider the set of required conditions (i.e. integrity constraints) that should be satisfied by any correct instance of the model. These conditions may be implicit in the model notation (like the multiplicity constraints in UML associations) or explicitly defined using a constraint language like OCL. These semantics problems may affect either the static (structural) or dynamic (behavioral) view of the system. Examples of possible errors are the non-executability of pre and postconditions in an operation or the presence of contradictory invariants or association multiplicities.

Different modeling notations and constraint languages have varying degrees of expressivity. Therefore, each notation creates a different challenge in terms of decidability and efficiency, and suggests a different set of analysis techniques. Due to space limitations, we will focus our discussion on the study of UML static models. In UML, static models may be expressed as class diagrams complemented with a set of OCL constraints.

A fundamental semantic correctness notion in static models is that of *model satisfiability.* (Strong) Satisfiability consists in deciding whether it is possible to create a non-empty and finite instantiation of the model in such a way that all integrity constraints are satisfied. Clearly, an unsatisfiable model is useless since every time users try to create a new object, e.g. instantiating one class of the model, at least one of the integrity constraints will become violated.

The importance of satisfiability comes from the ability to define many other correctness properties, such as liveliness, constraint redundancy, subsumption and so forth, in terms of the satisfiability problem. For example, a designer can check if an integrity constraint C is redundant by formulating a satisfiability problem where $\neg C$ replaces C in the model. If that model is satisfiable, it means it is possible to satisfy the remaining integrity constraints while violating C, so C is not a redundant constraint.

In the next sections we describe current approaches for UML/OCL model satisfiability and possible further research directions to cope with this challenging problem as a basis for identifying semantic defects in UML/OCL models.

3 State of the Art

In order to succeed in a MDD context, we believe any method for model satisfiability should fulfill the following list of requirements:

- Understand the input notation used by the designer (e.g. UML/OCL), not a formal notation nor a subset of that notation. If an internal formal notation is used, it should be transparent to the designer.
- Analyze the designer's model *as is*, without requiring any type of manual annotation.
- Perform the analysis automatically and without requiring user interaction.
- Provide results in a format meaningful to the designer.
- Be efficient and scale up to support large real-life examples.
- Integrate seamlessly into the designer tool chain.

Existing solutions lack of one or more of the previous qualities, and that might justify the lack of adoption of model-level verification tools in current MDD projects. In what follows, we describe the weaknesses of existing methods in terms of the main challenges they have to face when trying to satisfy the previous properties.

1. **Decidability:** The complexity of satisfiability analysis mainly depends on the expressiveness of the logic used to define the model and its constraints. Allowing a notation such as OCL makes the problem undecidable. Three different strategies are used to confront this undecidability:
   - Relaxing automation: Methods based on theorem proving might require user assistance during proofs, e.g. HOL-OCL [2].
   - Constraining the logic: Some methods work on a restricted subset of OCL (e.g. [3]) and some others do not support OCL at all (e.g. [4]). There is a trade-off between expressiveness (e.g. "are numerical constraints supported?") and complexity.
   - Performing bounded verification: If a finite bound is defined, it is then possible to check a property for all possible instances up to that maximum size, e.g. [5, 6]. This type of analysis can be used to prove the satisfiability of a model, but the lack of counterexamples within a bounded search space cannot be used to prove its unsatisfiability.
2. **Efficiency:** Reasoning on UML class diagrams is EXP-complete [7] even without OCL constraints and thus, current tools do not scale-up well which makes efficiency a concern for most non-trivial models.
3. **Usability:** Verification tools are often disappointing from the point of view of a designer. One of the main reasons is that tools do not directly manipulate the UML/OCL model but first translate it into a formal language (Alloy [6], CP [5], HOL [2], DL [8]) where the verification process takes place. Therefore, a good knowledge of this underlying language may be required to operate effectively, e.g. while selecting adequate parameters, tuning the model for the analysis or interacting with the tool. For this same reason, the interpretation of the results of the analysis might be complex and should be expressed in terms of the original model.

4. **Expressiveness:** The richness of the modeling languages (specially of the UML and OCL standards) creates a challenge for tool developers, who must support a wide variety of modeling constructs. Furthermore, the varied ecosystems of design tools, development tools and IDEs creates additional difficulties in terms of integration and interoperability.

4 Research Agenda: Promising Research Directions

The aim of this section is to sketch possible future research directions we believe may help in overcoming the previous limitations.

- **Automatic selection of the most appropriate verification approach for a specific model.** Each approach presents a different trade-off regarding the verification process. Depending on the model one approach may be more suited than others. For instance, for UML models without integrity constraints (a decidable problem) it may be better to use complete approaches (as those based on Description Logics) instead of approaches based on bounded verification.
- **Model partition to improve performance.** In most cases, the verification of a model m can be defined in terms of the verification of the submodels $m_i, \ldots, m_n$. Techniques for slicing the model in a subset of independent submodels (with the subsets to be computed depending on the property to be verified) will definitely help in improving the efficiency of the process due to its exponential nature.
- **Establish public community benchmarks to compare different tools and approaches.** Benchmarks provide an excellent resource to measure progress and the significance of a contribution. The existence of widely accepted benchmarks for model verification can foster progress and allow existing approaches to mature and exchange ideas.
- **Search space reduction for bounded methods.** Bounded methods require a finite search space. A smaller search space improves the efficiency but impairs the completeness of the verification. A preliminary analysis of the model could provide some insight on the best bounds of the search space as a trade-off between the two properties.
- **Apply SAT Modulo Theories to model satisfiability.** SAT Modulo Theories (SMT) is a promising technique for checking the satisfiability of a complex formula which combines recent improvements in SAT tools with the power of a custom solver specialised in a given logic [9]. In the case of model satisfiability, the challenge is identifying a subset of the modeling language which is sufficiently expressive yet allows efficient decision procedures.
- **Feedback improvements for defect correction.** Tools should not only be able to answer whether the model is correct. If the answer is no, the tool should be able to explain where, why and how it can be corrected.
- **Incremental verification.** The specification of a model is an iterative process where the model is continuously refined by means of adding, changing or

deleting some of its elements. Clearly, once a first version has been verified, we should be able to prove the correctness of new model versions without verifying the whole model again. Instead, only the “updated” parts should be considered.

– **Model normalization.** Normalizing a model, i.e. rewriting complex modeling constructs in terms of more basic ones, prior to the verification process helps to reduce the complexity of the verification algorithms that now do not need to consider the full language expressiveness. For instance, see [10] for some rules for normalizing OCL constraints.

5 Conclusions

Model-level verification is a key step toward improving software correctness. Despite the amount of research efforts devoted to this problem, existing approaches for model verification exhibit shortcomings that limit their applicability and adoption in MDE projects. In this paper, we have identified these problems and a possible set of research directions that may help in the creation of a new generation of verification tools that offer effectiveness, efficiency and usability.

References

1. Jones, C., O’Hearn, P., Woodcock, T.J.: Verified software: A grand challenge. IEEE Computer **39**(4) (2006) 93–95
2. Brucker, A.D., Wolff, B.: The HOL-OCL book. Technical Report 525, ETH Zurich (2006)
3. Queralt, A., Teniente, E.: Reasoning on UML class diagrams with OCL constraints. In Embley, D.W., Olivé, A., Ram, S., eds.: ER. Volume 4215 of Lecture Notes in Computer Science., Springer-Verlag (2006) 497–512
4. Baruzzo, A., Comini, M.: Static verification of UML model consistency. In Hearnden, D., S, J., Rapin, N., Baudry, B., eds.: 3rd Workshop on Model Design and Validation (MoDeV2a). (2006) 111–126
5. Cabot, J., Clarisó, R., Riera, D.: Verification of UML/OCL class diagrams using constraint programming. In: MoDeVVa 2008. ICST Workshop. (2008) available online: http://gres.uoc.edu/pubs/MODEVVA08.pdf
6. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: UML2Alloy: A challenging model transformation. In: ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems (MODELS 2007). Volume 4735 of Lecture Notes in Computer Science. (2007) 436–450
7. Berardi, D., Calvanese, D., Giacomo, G.D.: Reasoning on UML class diagrams. Artificial Intelligence **168** (2005) 70–118
8. van der Straeten, R., Mens, T., Simmonds, J., Jonckers, V.: Using description logic to maintain consistency between UML models. In: Proceedings of 6th International Conference UML 2003 - The Unified Modeling Language. (October 2003) 326–340
9. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT an SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM **53**(6) (November 2006) 937–977
10. Cabot, J., Teniente, E.: Transformation techniques for OCL constraints. Science of Computer Programming. **68**(3) (2007) 179–195