MaxUSE: A Tool for Finding Achievable Constraints and Conflicts for Inconsistent UML Class Diagrams

Hao Wu

Department of Computer Science,
National University of Ireland, Maynooth
haowu@cs.nuim.ie

Abstract. In the context of Model Driven Engineering (MDE), the structure of a system is typically described by using a UML class diagram annotated with a set of Object Constraint Language (OCL) constraints. These constraints specify rules that are not expressible by using structural features. These constraints can be conflicting, resulting in inconsistencies. When this happens, the existing tools terminate and provide no information about which constraints are achievable and which ones cause conflicts. In this paper, we present MaxUSE, a tool for finding achievable OCL constraints and conflicts for inconsistent UML class diagrams. MaxUSE integrates the USE modeling tool with a satisfiability modulo theories (SMT) solver. It finds a set of achievable constraints based on their rankings by casting to a weighted MaxSMT problem and at the same time locates constraint conflicts. We use an example to demonstrate MaxUSE's usage scenarios and discuss its usefulness to the community.

1 Introduction

Model-Driven Engineering (MDE) plays a significant role in modern software development by exploiting different abstract models. Among them, Unified Modeling Language (UML) is a common modeling language for modeling a system at an abstract level. It uses structure and behaviour diagrams to depict static and dynamic aspects of a system. For example, using class diagrams to model relationships between different entities and state machines to capture possible transitions from one state to another. On other hand, Object Constraint Language (OCL), a declarative language, is used to describe necessary rules that can not be expressed by UML diagrams. These rules impose additional constraints over different structural features to eliminate undesirable scenarios. Verifying consistency of a UML model therefore becomes a task of finding an *instance* that conforms to not only structural constraints but also OCL constraints.

Recently, a number of tools and approaches have been proposed to verify the consistency of a UML class diagram by employing formal verification techniques [1–4]. However, when a UML class diagram is inconsistent these tools typically have no knowledge about the constraints that cause conflicts. Knowing information about which constraints cause conflicts is very helpful for users

to understand and fix their class diagrams. In practice, users may also wish to know how many constraints can be achieved in the current diagram, and use this information for further refining their class diagrams. For example, a user may be interested in fixing the minimum number of constraints that cause conflicts. In other scenarios, users may treat individual constraints differently based on their own domain-specific knowledge and look for an instance that satisfies the most important OCL constraints.

In this paper, we present MaxUSE, an automated tool for finding the set of achievable constraints based on user's rankings and constraint conflicts for inconsistent UML class diagrams. MaxUSE extends USE, an existing modeling tool, by integrating an SMT solver as its back-end reasoning engine. It finds the maximum total rank by solving a weighted MaxSMT problem and identify constraint conflicts by solving the set cover problem. Detailed theories and algorithms have been addressed in [5]. Here, we describe the integration of a modeling tool with an SMT solver (Section 2) and demonstrate MaxUSE's usage scenarios (Section 3) by illustrating it with an example.

2 Overall Architecture

MaxUSE is built on top of the USE modeling tool. It exploits USE's front-end to read in a UML class diagram annotated OCL constraints and generates SMT assertions that can be solved by an SMT solver. The overall architecture of MaxUSE, as shown in Figure 1, consists of three layers: USE, Uran and Solver.

USE. USE is an open-source modeling tool that allows users to construct UML class diagrams in its own specification [6]. It also supports constraints written in OCL. USE provides a set of commands that enable users to construct object diagrams (instances) and check whether an object diagram (instance) conforms to its class diagram's structural and OCL constraints. To support ranked constraints, we change USE's front-end by modifying its grammars, UML and OCL metamodels (abstract syntax trees). We then implement two visitors that traverse and store each model feature, such as an association and a class invariant, into a temporal memory location that can be used by Uran.

Uran. Uran is an open-source project that aims to provide users with an engine for constructing and evaluating SMT2 assertions through well-defined APIs [1]. The purpose of Uran is to decouple assertion generation functionalities from modules that are designed for other purposes. This design allows users to freely modify and upgrade assertion generation to accommodate specific purposes without affecting other modules. Further, Uran directly interacts with an SMT solver via different APIs. Currently, Uran is able to communicate with the Z3 SMT solver. MaxUSE uses Uran API's to translate model features extracted from a USE specification to a set of predicates, functions and objects. It then outputs a set of well-formed SMT2 assertions associated with corresponding ranks. In other words, it formalises a UML class diagram with ranked OCL constraints into a weighted MaxSMT problem.

[1] available at: https://github.com/classicwuhao/uran

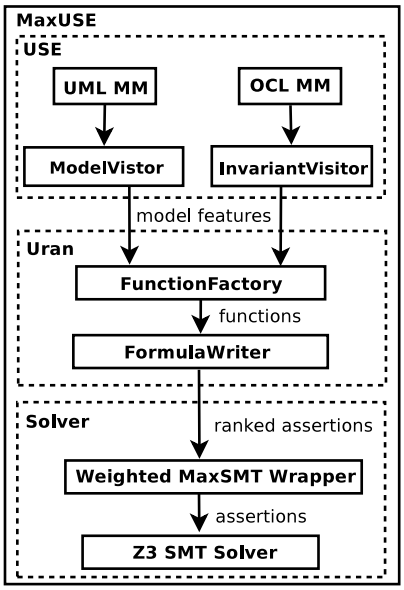


Fig. 1. The architecture of MaxUSE integrates with three layers: USE, Uran and Solver.

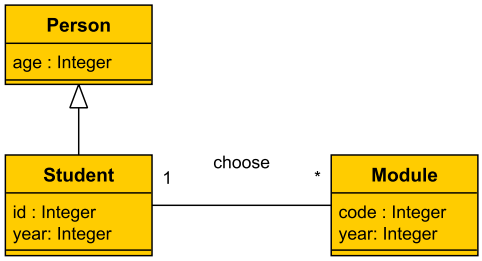
Solver. In order to solve this weighted MaxSMT problem, we implement a wrapper that iteratively calls the Z3 SMT solver until we find the optimal value. To reduce the number of calls to the solver, this wrapper uses a binary-search based algorithm to find the optimal value. Once an optimal values is found, we enumerate all possible ways of achieving this value by blocking all previous successful assignments until no more assignments can be found. Each assignment found by the solver is a weighted MaxSMT solution. Finally, we map each solution back to a corresponding model feature and generate a report. To find all constraint conflicts, we formalise the set of weighted MaxSMT solutions into the set cover problem and solve it using the algorithm described in [5]. Therefore, MaxUSE uses an SMT solver for computing both sets of achievable constraints and conflicts for inconsistent UML class diagrams.

3 Usage Scenarios

In this section, we illustrate three usage scenarios of using MaxUSE. The three scenarios discussed in this Section are based on the example shown in Figure 2. This example uses a UML class diagram to represent a real world example of students in a university choosing multiple modules to study. This class diagram is enriched with 8 OCL constraints specified as class invariants ($inv1$ to $inv8$) under three classes. For example, each student must have a unique id number ($inv4$) and can only choose modules that are in their year ($inv5$). In this example, we use numbers 1 to 6 to distinguish a student's year, and students that are in

year 6 are considered as research students. Thus, a university has some non-research and research students ($inv6$). All invariants except for $inv8$ are ranked by using an integer value.

In this example, 7 ($inv1$-$inv7$) out of 8 class invariants are ranked. We consider a ranked class invariant as a *soft constraint*. This means that it might be switched off during the search for the maximum total rank. For example, $inv4$ is a soft constraint and is ranked as 5. On the other hand, if an invariant is *not* ranked, then it is a *hard constraint* that must not be ignored during the search. For example, $inv8$ in Figure 2 must hold, no matter what. Therefore, this allows users to rank OCL constraints in a UML class diagram in 3 different ways: (1) not ranked at all (hard constraints only) (2) totally ranked (soft constraints only) (3) partially ranked (a mixture of soft and hard constraints).




```
context Person
@Rank = 2
inv1: Person.allInstances()->exists(p|p.age>0 and p.age<18)

context Student
@Rank = 3
inv2: self.age>18
@Rank = 4
inv3: self.year>=1 and self.year<=6
@Rank = 5
inv4: Student.allInstances()->forAll(s1,s2:Student|s1<>s2 implies s1.id <> s2.id)
@Rank = 6
inv5: Student.allInstances()->forAll(s|s.modules->forAll(m|s.year=m.year))
@Rank = 6
inv6: Student.allInstances()->exists(s|s.year=6) and Student.allInstances()->exists(s|s.year<6)
@Rank = 7
inv7: Student.allInstances()->forAll(s|s.modules->notEmpty())

context Module
inv8: self.year>=1 and self.year<=5
```

Fig. 2. A UML class diagram annotated with ranked OCL constraints. The ranks are highlighted in the shaded area.

3.1 Verifying Consistency

If the set of OCL constraints is not ranked at all, then this means that every single constraint must hold. In this scenario, MaxUSE translates a UML class diagram

with its OCL constraints to a set of SMT assertions. A UML class diagram is consistent iff generated SMT assertions are satisfiable. MaxUSE uses the Z3 SMT solver to determine the satisfiability of these assertions. In our example, MaxUSE is unable to find an instance that satisfy all 8 OCL constraints in Figure 2, assuming that all constraints are hard constraints. This is due to 2 conflicts (sets): ($inv1$,$inv2$) and ($inv5$,$inv6$,$inv7$,$inv8$). In other words, removal of any elements in a conflict (set) makes the remaining elements achievable. For example, $inv5$, $inv7$ and $inv8$ are achievable if $inv6$ is removed from the class diagram.

3.2 Finding Achievable Constraints

In many practical situations, users may treat individual constraints differently. For example, a university may consider a registration procedure where students choosing some modules ($inv7$) is more important than choosing modules in their corresponding year ($inv5$). We thus allow users to freely rank individual constraints to distinguish their importance. In this scenario, MaxUSE calculates a total rank from the set of soft constraints and computes a maximum achievable rank by solving a weighted MaxSMT problem.

In Figure 2, 7 ranked class invariants result in a total rank of 33. Since this UML class diagram is inconsistent, MaxUSE maximises this total rank up to 25. In fact, MaxUSE finds a total of two solutions that can achieve this value. These two solutions are listed in Table 1.

Solution 1		Solution 2	
Invariant	Rank	Invariant	Rank
inv1	**0**	**inv1**	**0**
$inv2$	3	$inv2$	3
$inv3$	4	$inv3$	4
$inv4$	5	$inv4$	5
$inv5$	6	**inv5**	**0**
inv6	**0**	$inv6$	6
$inv7$	7	$inv7$	7
$inv8$	NA	$inv8$	NA

Table 1. Two solutions that can achieve a maximum rank of 25. Each solution contains a set of 5 achievable invariants out of 7 soft constraints. The invariants that cannot be met are marked with 0 in the "Rank" column. "NA" indicates that a corresponding class invariant is a hard constraint.

MaxUSE finds a maximum of 5 achievable invariants out of 7 soft constraints in Figure 2. Note that MaxUSE always first verifies the consistency of a UML class diagram. If the UML class diagram is consistent, then MaxUSE terminates since every constraint is achievable. In other words, MaxUSE finds the set of achievable (ranked) constraints only when a UML class diagram is *not* consistent.

3.3 Finding Constraint Conflicts

To find *all* conflicts among OCL constraints, MaxUSE first treats each constraint equally, then casts it to a MaxSMT problem [2] and solves it by using the Z3 SMT solver. Here, the returned solutions to MaxSMT is a set, namely they are MaxSMT solutions. Each MaxSMT solution in this set represents a way of achieving a maximum *number* of constraints. MaxUSE formalises this set of solutions into the set cover problem and solves it by using the algorithm in [5]. This algorithm is inspired by the work on using the set cover problem to model conflicts among SAT formulas [7]. Finally, MaxUSE interprets each solution to the set cover problem as a conflict.

For the class invariants in Figure 2, MaxUSE finds a total of 8 possible ways of achieving a maximum number of 6 class invariants (shown in Table 2) and 2 conflicts: ($inv1$,$inv2$) and ($inv5$,$inv6$,$inv7$,$inv8$).

	$Inv1$	$Inv2$	$Inv3$	$Inv4$	$Inv5$	$Inv6$	$Inv7$	$Inv8$
(1)	✓	✗	✓	✓	✗	✓	✓	✓
(2)	✓	✗	✓	✓	✓	✗	✓	✓
(3)	✓	✗	✓	✓	✓	✓	✗	✓
(4)	✓	✗	✓	✓	✓	✓	✓	✗
(5)	✗	✓	✓	✓	✗	✓	✓	✓
(6)	✗	✓	✓	✓	✓	✗	✓	✓
(7)	✗	✓	✓	✓	✓	✓	✗	✓
(8)	✗	✓	✓	✓	✓	✓	✓	✗

Table 2. A total of 8 MaxSMT solutions. Each one of them represents a way of achieving a maximum 6 number of class invariants shown in Figure 2. We use a ✓ to indicate an invariant is achievable and a ✗ to denote an invariant that cannot be met.

The first conflict is quite obvious and it is caused by the invariants $inv1$ and $inv2$ defined for the *age* attribute. However, the second conflict is not easy to identify. This conflict is caused by the invariants that there must exist some research and non-research students ($inv6$) choosing some modules ($inv7$) in their corresponding year ($inv5$). However, there are modules that are only available for non-research students ($inv8$: between year 1 and 5).

4 Usefulness

By integrating an SMT solver into a modeling environment, users are now able to use MaxUSE to tackle ranked OCL constraints in a UML class diagram. More importantly, when a UML class diagram is not consistent users no longer need to spend time on working out which constraints are achievable and which ones cause the conflicts. They can easily use MaxUSE to find out this information. In practice, this is a very effective and efficient way for further refining class

[2] Note that since the rank for each constraint is equal, this means that a weighted MaxSMT can be treated as a MaxSMT problem.

diagrams. Further, our evaluation results in [5] suggest that MaxUSE scales reasonably well and the quality of computed constraint conflicts is high. Therefore, we believe that users from the software verification and Model Driven Engineering community can benefit from its capabilities.

5 Availability

MaxUSE is a free and open-source project hosted on GitHub under the GNU public license:

https://github.com/classicwuhao/maxuse

The repository is accompanied with detailed instructions and examples that show how to build and use MaxUSE. The implementation of MaxUSE consists of approximately Java 7000 lines of code. Currently, MaxUSE is command based and easy to install using the provided build script. In addition, the benchmark that we used for evaluating MaxUSE is also available in the repository.

6 Related Work

Even though a number of tools or approaches leverage the power of constraint solvers and theorem provers for verifying/reasoning UML models, they do not support ranked constrains and conflict finding [8, 9, 1–3, 10–15, 4, 16]. To the best of our knowledge, MaxUSE is the first automated tool that supports finding achievable constraints (based on rankings) and conflicts for UML class diagrams.

The USE modeling tool takes a similar apporach to UML2Alloy. It integates with a relational model finder for verifying UML class diagrams [1, 17]. However, the encodings used in the model finder are limited to boolean formulas, and thus they are not suitable for tackling numeric constraints. In particular, numeric ranks are defined for each OCL constraint. UMLtoCSP verifies EMF models by casting it to a Constraint Satisfaction Problem (CSP) [15, 18–20]. However, they only allow users to check weak and strong satisfiability by generating a different number of instances for every class. The HOL-OCL tool encodes OCL into the Higher-order Logic (HOL) and uses Isabelle to reason about UML class diagrams [13]. Since Isabelle is an interactive theorem prover, the level of automation is quite limited and the feedback can be difficult to interpret by software engineers.

7 Conclusion

In this paper, we demonstrate how MaxUSE integrates an SMT solver into a modeling environment. This integration allows users to leverage efficient SMT solving to reason over ranked constraints defined in a UML class diagram. In addition, MaxUSE can significantly reduce the amount of effort in investigating inconsistencies in UML class diagrams by automatically finding the set of achievable OCL constraints and conflicts. In the future, we plan to build a plug-in for MaxUSE to allow us to exploit multiple SMT solvers for reasoning over a considerably large number of OCL constraints.

References

1. Kuhlmann, M., Hamann, L., Gogolla, M.: Extensive validation of OCL models by integrating SAT solving into USE. In: 49th TOOLS, Zurich, Switzerland, Springer (2011) 290–306
2. Wille, R., Soeken, M., Drechsler, R.: Debugging of inconsistent UML/OCL models. In: 2012 DATE. (2012) 1078–1083
3. Wu, H., Monahan, R., Power, J.F.: Exploiting attributed type graphs to generate metamodel instances using an SMT solver. In: 7th TASE, Birmingham, UK (2013)
4. Dania, C., Clavel, M.: Ocl2msfol: A mapping to many-sorted first-order logic for efficiently checking the satisfiability of OCL constraints. In: 19th MoDELS, ACM (2016) 65–75
5. Wu, H.: Finding achievable features and constraint conflicts for inconsistent metamodels. In: 13th ECMFA. (2017)
6. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-based specification environment for validating UML and OCL. Science of Computer Programming **69**(1-3) (2007) 27–34
7. Liffiton, M.H., Sakallah, K.A.: Algorithms for computing minimal unsatisfiable subsets of constraints. J. Autom. Reason. **40**(1) (Janurary 2008) 1–33
8. Beckert, B., Keller, U., Schmitt, P.H.: Translating the Object Constraint Language into first-order predicate logic. In: FLoC @ 3rd Federated Logic Conferences,Denmark (2002)
9. Maraee, A., Balaban, M.: Efficient reasoning about finite satisfiability of UML class diagrams with constrained generalization sets. In: ECMDA, Springer (2007) 17–31
10. Soeken, M., Wille, R., Drechsler, R.: Encoding OCL data types for SAT-based verification of UML/OCL models. In: 5th TAP, Zurich, Switzerland, Springer (2011) 152–170
11. Büttner, F., Egea, M., Cabot, J.: On verifying ATL transformations using 'off-the-shelf' SMT solvers. In: 15th MoDELS. (2012) 432–448
12. Clavel, M., Egea, M., de Dios, M.A.G.: Checking unsatisfiability for OCL constraints. ECEASST **24** (2009)
13. Brucker, A.D., Wolff, B.: HOL-OCL – A Formal Proof Environment for UML/OCL. In: The 11th FASE, Springer (2008) 97–100
14. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: UML2Alloy: A challenging model transformation. In: ACM/IEEE 10th MoDELS, Nashville, TN, Springer (2007) 436–450
15. Beckert, B., Hähnle, R., Schmitt, P.H.: Verification of Object-oriented Software: The KeY Approach. Springer, Berlin, Heidelberg (2007)
16. Wu, H.: Generating metamodel instances satisfying coverage criteria via SMT solving. In: The 4th MODELSWARD. (2016) 40–51
17. Torlak, E., Jackson, D.: Kodkod: a relational model finder. In: 13th TACAS, Braga, Portugal, Springer (2007) 632–647
18. González Pérez, C.A., Buettner, F., Clarisó, R., Cabot, J.: EMFtoCSP: A tool for the lightweight verification of EMF models. In: SEMF:Rigorous and Agile Approaches, Zurich, Suisse (2012)
19. Cabot, J., Clarisó, R., Riera, D.: Verification of UML/OCL class diagrams using constraint programming. In: IEEE STV&V Workshop, Berlin, Germany, IEEE Computer Society (2008) 73–80
20. Cabot, J., Clarisó, R., Riera, D.: On the verification of UML/OCL class diagrams using constraint programming. Journal of Systems and Software **93** (2014) 1–23