Science of Computer Programming 228 (2023) 102955

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Original software publication

QMaxUSE: A new tool for verifying UML class diagrams and OCL invariants

Hao Wu

Computer Science Department, Maynooth University, Ireland

ARTICLE INFO

Article history:
Received 31 May 2022
Received in revised form 28 February 2023
Accepted 4 April 2023
Available online 6 April 2023

Keywords:
Verification
UML&OCL
Query SMT
Unsatisfiable core

ABSTRACT

Formal verification of a UML class diagram annotated with OCL constraints has been a long-standing challenge in Model-driven Engineering. In the past decades, many tools and techniques have been proposed to tackle this challenge. However, they do not scale well and are often unable to locate the conflicts when then number of OCL constraints significantly increases. In this paper, we present a new tool called QMaxUSE. This tool is designed for verifying UML class diagrams annotated with large number of OCL invariants. QMaxUSE is easy to install and deploy. It offers two distinct features. (1) A simple query language that allows users to choose parts of a UML class diagram to be verified. (2) A new procedure that is capable of performing concurrent verification.

Code metadata

Code metadata description	
Current code version	1.0.3
Permanent link to code/repository used for this code version	https://github.com/ScienceofComputerProgramming/SCICO-D-22-00134
Permanent link to Reproducible Capsule	https://github.com/classicwuhao/qmaxuse
Legal Code License	GNU General Public License (GPL)
Code versioning system used	git
Software code languages, tools, and services used	Java 1.7, USE Modelling Tool, Z3
Compilation requirements, operating environments and dependencies	Microsoft Windows, Linux, MacOS.
If available, link to developer documentation/manual	https://github.com/classicwuhao/qmaxuse
Support email for questions	haowu@cs.nuim.ie

1. Introduction

In software engineering, Unified Modeling Language (UML) class diagrams along with Object Constraint Language (OCL) are typically used to model structures of a system [1,2]. For example, an entity of a system is depicted as a class, and relationships between different entities are represented as inheritance or associations. OCL is then used by modeling practitioners to express additional constraints that cannot be captured by the UML graphical notation. For example, a user may impose a constraint over an attribute of a class diagram.

However, reasoning or verifying the consistency of a UML class diagram is quite challenging [3,4]. Formally, verifying the consistency means checking whether a valid instance can or cannot be generated from a UML class diagram and its OCL

E-mail address: haowu@cs.nuim.ie.

https://doi.org/10.1016/j.scico.2023.102955

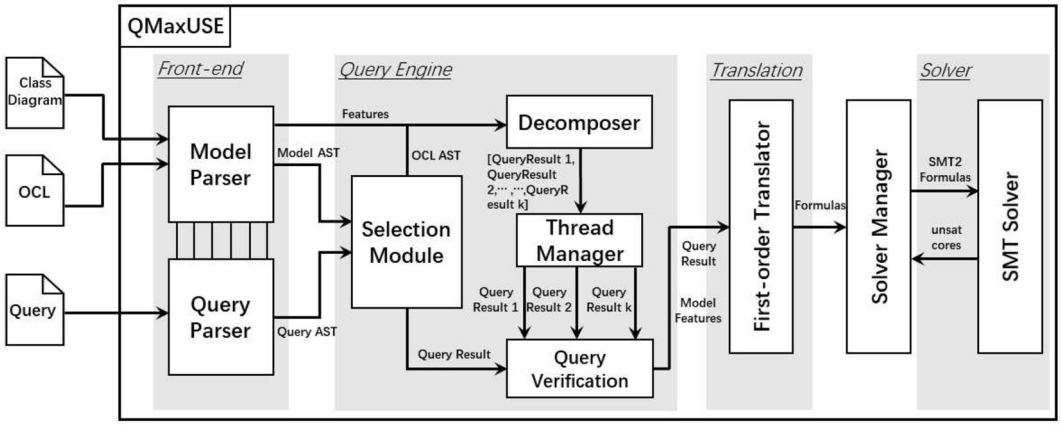


Fig. 1. The overall architecture of QMaxUSE.

constraints. If it cannot be generated (found), then a UML class diagram is said to be inconsistent.[1] This implies that there exists at least one conflict in the structural and OCL constraints defined in the UML class diagram. To tackle this challenge, many approaches and techniques have been proposed [5–11]. There are still two main challenges remaining: (1) When a UML class diagram is inconsistent, many existing tools are unable to pinpoint the set of OCL constraints that cannot be satisfied. (2) When the number of OCL constraints significantly increases, the existing tools and techniques do not scale well [9,12].

The performance and scalability of a tool is particularly important for industries [13–15]. This is because they typically have models with a large number of OCL constraints. Hence, our aim in this paper is to present our latest verification tool QMaxUSE [16,17] that has the following two distinct features:

1. Provide an interactive verification that allows users to incrementally verify the consistencies of their models by selecting different parts of their design for analysis.
2. Offer a scalable approach that can pinpoint the conflicting constraints when a model has a large number of complex OCL invariants.

2. Architecture

QMaxUSE is very easy to install, use and no additional libraries are required. QMaxUSE is written in Java and consists of about 33k lines of code. Currently, QMaxUSE supports Windows, Linux and macOS. QMaxUSE is a command-line tool based verification tool and uses the same USE modelling tool GUI for showing pre-defined queries. Fig. 1 shows the overall architecture of QMaxUSE. It has four main layers: front-end, query engine, translation and solver. Each layer has its own functionality and provides relevant information for the next layer. We now describe each layer in detail.

Front-end. This layer is responsible for parsing queries issued by users and generating query ASTs (abstract syntax trees) along with information collected from a UML class diagram and OCL invariants. To successfully parse a query from a user, we have designed a query parser that extends the existing USE's parser. Our query grammar consists of about 14 rules that are written in ANTLR specification. During the traversal of a query, our parser constructs a corresponding query AST and checks its semantics including query operators, types of a set of features and integrity of OCL invariants. Our parser reports syntax and semantics errors to users if there is any.

Query Engine. This layer has four main modules: selection, decomposition, thread management and query verification.

1. **Selection Module:** The selection module uses a selection algorithm to traverse the ASTs generated from the font-end layer and produces a query result. In particular, our selection algorithm visits each node of a query AST, analyses the type of each operator and collects different model features from a model. The collected model features are saved in a *query result* that essentially is a set. This set contains a list of classes, attributes, associations and OCL invariants to be verified. The syntax and semantics of our query language is described in [18].
2. **Decomposer:** The decomposition module has a specialised algorithm that is able to decompose a class diagram along with OCL invariants into a set of different queries. These queries then can be verified concurrently using a query verification procedure. This module is enabled when a user would like to perform a concurrent verification. With the support of this decomposition algorithm, it is now possible to verify a UML class diagram that has a large number of OCL invariants. The algorithms used in this module are described in [18].

[1] Note that for this scenario, this may also mean that an algorithm may not terminate and the problem itself is not decidable.

3. **Thread Manager:** This module is responsible for controlling the number of threads to be spawned for verification. Depending on configurations, users could specify a particular number of threads for concurrent verification for both simpler and complex verification tasks. By default, the number of threads created is based on the number of queries produced by our decomposition module (Decomposer).
4. **Query Verification:** This module uses an algorithm to take care of the verification of a single query. In other words, when a user issues a query in QMaxUSE, this module formally verifies the parts touched by the query.

Translation. This layer uses a *first-order translator* to translate a query into a set of first-order formulas that can be verified by the SMT solver. The translation here is similar to the one described in [9]. We use first-order formulas to encode classes or attributes and linear integer inequalities to capture the multiplicities at an association-end. For an OCL invariant, we traverse its AST and generate an SMT formula by using a combination of first-order theories.
Solver. At this layer, we design a new interface (*SolverManager*) to reduce the interaction overhead between the translation and solver layer. This interface is able to directly interact with an SMT solver and can be extended to multiple SMT solvers. Currently, QMaxUSE uses Z3 as its default SMT solver [19].[2]

2.1. Software functionalities

We describe the functionalities QMaxUSE offers to facilitate query and verification.

- **Feature Selection.** A user can issue a query to select features of a UML class diagrams. Every query must use a *select* statement. The features can be selected by a query are: attributes, classes, associations and OCL invariants. A wildcard character * can be used within a select statement to choose many features. The rule here is when a feature is selected, its owner is also implicitly selected. For example, the following query

  ```
  select Person.* with Student::*
  ```

  selects all attributes of the *Person* class and invariants of the *Student* class, and the classes *Person* and *Student* are also implicitly selected.
- **Joint Query.** Multiple queries can be joined together as a *joint query*. To form a joint query, a user can use one of the three joint operators: intersection (&), union (+) and difference (-). To facilitate forming multiple complex queries into a joint query, QMaxUSE provides users with an *alias* expression. This allows users to refer an alias in a joint query. For example, the following two queries

  ```
  select Person.* with Student::*, Student::* as q1
  select University.*, Module.* as q2
  ```

  use two alias names $q1$ and $q2$ respectively. A joint query (intersection) can then be easily formed by using `q1 & q2` to select common features shared by these two queries.
- **Pre-defined Query.** QMaxUSE provides a *query module* that allows users to store their pre-defined queries so that they can be used as test cases to cover parts of their UML class diagrams. A query module is saved in a USE specification file and automatically loaded when a user launches QMaxUSE.
- **Concurrent Verification.** A user can use command `qverify` in the QMaxUSE command-line to start a concurrent verification on current loaded UML class diagram. This command decomposes a class diagram into many smaller queries and verifies each of them concurrently. QMaxUSE lists relevant conflicting OCL invariants if a query contains any.

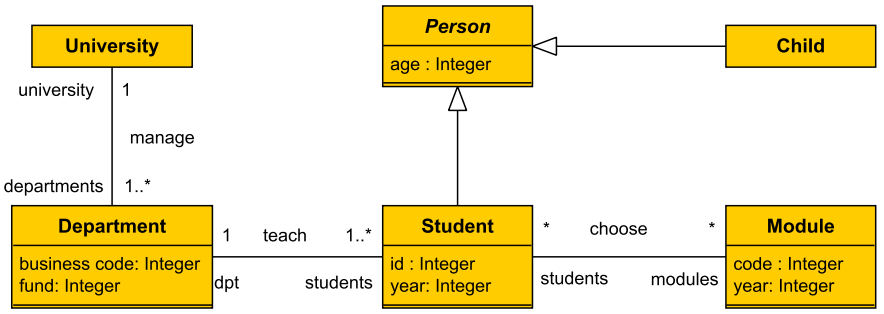
3. An illustrative example

In this section, we provide an illustrative example to show how to use QMaxUSE to verify a UML class diagram. This example was first introduced in [20,12] and later extended in [9]. It models a university student that can select multiple modules to study. This is shown in Fig. 2. Furthermore, this UML class diagram has 8 OCL class invariants. These invariants impose additional constraints. For example, $inv5$ indicates a student can select modules that are available only in their year.[3] A department must have some research students and non-research students. This constraint is reflected in $inv6$. However, this class diagrams has a total number of two conflicting OCL invariants.

A user can use QMaxUSE to verify this UML class diagram in two ways: query and concurrent verification. For queries-based verification, a user may issue a query (in QMaxUSE) that covers a part of this diagram, and QMaxUSE is able to verify the parts covered by the query. For example, a user could issue the following query to select the attributes from the **Person** and **Student** class along with two OCL invariants: $inv1$ and $inv2$.

```
select Person.*, Student.* with Person::inv1, Student::inv2
```

[2] Though users can switch to different SMT solvers, the completeness of algorithms for extracting unsat cores is determined by individual SMT solver.
[3] In this example, numbers 1 to 6 are used to distinguish a student's year. Students that are in year 6 are considered as research students.




```
context Person
inv1: Person.allInstances()->forAll(p|p.age>0 and p.age<18)


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

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

Fig. 2. A UML class diagram with the 8 OCL class invariants shows how the students in each department can choose multiple modules to study.

Fig. 3. A screenshot showing the verification result returned from QMaxUSE for a single query.

QMaxUSE verifies the features collected by this query and reports a conflict between $inv1$ and $inv2$. This is because $inv1$ indicates that every person's age is between 0 and 18, while $inv2$ requires that every student in a university is over 18. Hence, it is impossible to create an instance of the **Student** class. A screenshot of the verification result of the query is shown in Fig. 3.

To enable concurrent verification, a user can just simply type $qverify$ to launch our decomposition algorithm. This algorithm creates a number of threads and each of them verifies a part of the class diagram. In this example, a number of 3 threads are created. This is because this class diagram can be decomposed into 3 different queries. A screenshot of concurrent verification for the class diagram (Fig. 2) is shown in Fig. 4. The second conflict implies that a department allows some research and non-research students ($inv6$) to choose some modules ($inv7$) in their corresponding year ($inv5$). However, $inv8$ indicates that modules are only available for non-research students ($inv8$: between year 1 and 5).

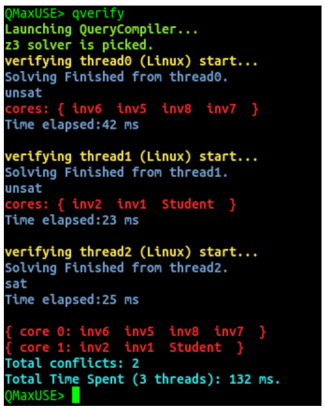


Fig. 4. A screenshot showing QMaxUSE's concurrent verification results for the UML class diagram in Fig. 2.

Table 1
Evaluation results. Invs=number of OCL invariants, Nodes=size of invariant ASTs, Quant=number of quantifiers, Op=number of operators. TO= Timeout (20min), MaxUSE=our previous tool without query and concurrent verification support.

	Name	OCL Structure Size				MaxUSE [12]	QMaxUSE	
		Invs	Nodes	Quant	Op	Time (sec)	Threads	Time (sec)
Part A	A4	8	73	7	18	0.204	3	0.241
	B1	27	150	10	30	4.528	23	2.022
	B2	45	266	13	57	56.846	32	3.046
	C1	29	201	24	33	38.167	19	3.413
	C2	43	279	28	51	91.319	33	5.954
Part B	B4	90	599	23	152	159.378	68	7.151
	B5	136	925	44	228	TO	90	118.64
	C5	156	1008	100	184	TO	90	114.41
	D5	166	1143	131	225	TO	95	14.026
	E4	105	985	56	246	TO	42	4.464
	E5	167	1134	68	325	TO	45	3.653

4. Evaluation results

We use a benchmark from [9] to show the size and the complexities of OCL invariants QMaxUSE can handle. This benchmark has a total of 25 different class diagrams annotated with different size of OCL invariants. This benchmark covers a wide range of OCL language features including: nested quantifiers, collections, logical/arithmetic operations and navigations.

Table 1 summarises a subset of our evaluation results for QMaxUSE.[4] Part A shows QMaxUSE' performance on smaller size OCL invariants while Part B demonstrates its performance on large size OCL invariants. The more detailed evaluation results including a comparison against other tools can be seen in [17]. The evaluation shown here is carried out on an Intel(R) Core (TM) machine that has six 2.8 GHz cores with 16 GB memory. The underlying SMT solver is the Z3 SMT solver (version 4.8.10).

In general, QMaxUSE can significantly improve verification performance. This is mainly due to two reasons. (1) We directly extract unsat cores from the SMT solver (Z3) through its dedicated APIs. This is fundamentally different from our previous tool MaxUSE [12,9]. The algorithm MaxUSE tries to satisfy as many OCL invariants as possible by reducing to a weighted MaxSMT problem, and then pinpoints conflicting invariants by solving the set cover problem [21]. Although this algorithm is generic, it can easily become very slow when the number of formulas increases. QMaxUSE operates differently from MaxUSE. It directly extracts unsat cores from the solver without implementing an extra layer on top of the solver. However, the limitation here is that a solver must support unsat core extraction. (2) Our concurrent verification algorithm is able to generate a number of much smaller size of formulas. These formulas can be scheduled to be verified efficiently even via a single thread since they are much less complex and smaller. This can significantly improve QMaxUSE's performance and makes it possible to verify a large number of OCL invariants. We observe that the number of nested quantifiers in a formula ($\forall\ \exists$) can pose a significant challenge to the solver. Therefore, a verification engine that relies on the solvers may have the same challenge especially when the number of nested formulas dramatically increases. In a different manner, QMaxUSE is able to shift solving these large size of formulas to runs on a much smaller size of formulas. This yields a much better result and performance.

[4] More detailed evaluation including comparison is in [18].

5. Conclusion

Though many existing tools and techniques have been proposed for verifying UML class diagrams annotated with OCL invariants, they often do not scale well and unable to pinpoint conflicts. QMaxUSE offers an elegant solution to scalability problem by providing query-based verification. This allows users to issue queries that can cover different parts of a class diagram. It also can decompose a class diagram that has a large number of OCL invariants into different much smaller queries for verification. In the future, we plan to extend QMaxUSE to support OCL operational contracts so that users can reason about dynamic aspects of a system. The challenge here is to integrate our current technique into a specification language that can capture dynamic behaviour of a system.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] C. Atkinson, T. Kühne, Model-driven development: a metamodeling foundation, IEEE Softw. 20 (5) (2003) 36–41.
[2] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, second edition, Addison-Wesley Professional, 2005.
[3] D. Berardi, D. Calvanese, G.D. Giacomo, Reasoning on UML class diagrams, Artif. Intell. 168 (1–2) (2005) 70–118.
[4] A. Queralt, E. Teniente, Reasoning on UML class diagrams with OCL constraints, in: Conceptual Modeling, Springer, 2006, pp. 497–512.
[5] M. Gogolla, F. Hilken, K. Doan, Achieving model quality through model validation, verification and exploration, Comput. Lang. Syst. Struct. 54 (2018) 474–511.
[6] A. Queralt, A. Artale, D. Calvanese, E. Teniente, OCL-Lite: finite reasoning on UML/OCL conceptual schemas, Data Knowl. Eng. 73 (2012) 1–22.
[7] A. Maraee, M. Balaban, Efficient reasoning about finite satisfiability of UML class diagrams with constrained generalization sets, in: 3rd European Conference Model Driven Architecture, Springer, 2007, pp. 17–31.
[8] M. Balaban, A. Maraee, Finite satisfiability of UML class diagrams with constrained class hierarchy, ACM Trans. Softw. Eng. Methodol. 22 (3) (2013) 24.
[9] H. Wu, M. Farrell, A formal approach to finding inconsistencies in a metamodel, in: Software and Systems Modeling, Springer, 2021.
[10] M. Kuhlmann, M. Gogolla, Strengthening SAT-Based Validation of UML/OCL Models by Representing Collections as Relations, Modelling Foundations and Applications, vol. 7349, Springer, 2012, pp. 32–48.
[11] O. Semeráth, A. Vörös, D. Varró, Iterative and incremental model generation by logic solvers, in: 19th International Conference on Fundamental Approaches to Software Engineering, Springer, 2016, pp. 87–103.
[12] H. Wu, MaxUSE: a tool for finding achievable constraints and conflicts for inconsistent UML class diagrams, in: 13th International Conference on Integrated Formal Methods, Springer, 2017, pp. 348–356.
[13] S. Ali, T. Yue, M. Zohaib Iqbal, R.K. Panesar-Walawege, Insights on the use of OCL in diverse industrial applications, in: D. Amyot, P. Fonseca i Casas, G. Mussbacher (Eds.), System Analysis and Modeling: Models and Reusability, Springer International Publishing, Cham, 2014, pp. 223–238.
[14] M.Z. Iqbal, S. Ali, T. Yue, L. Briand, Experiences of applying UML/MARTE on three industrial projects, in: R.B. France, J. Kazmeier, R. Breu, C. Atkinson (Eds.), Model Driven Engineering Languages and Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 642–658.
[15] D. Garry, T. Balfe, Experiences using OCL for business rules on financial messaging, in: Proceedings of the 12th Workshop on OCL and Textual Modelling, OCL '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 65–66.
[16] H. Wu, QMaxUSE: a query-based verification tool for UML class diagrams with OCL invariants, in: 25th International Conference on Fundamental Approaches to Software Engineering, Springer, Munich, Germany, 2022.
[17] H. Wu, A query-based approach for verifying UML class diagrams with OCL invariants, in: 18th European Conference on Modelling Foundations and Applications, 2022.
[18] H. Wu, A query-based approach for verifying UML class diagrams with OCL invariants, in: The 18th European Conference on Modelling Foundations and Applications, ECMFA 2022, J. Object Technol. 21 (3) (2022) 1–17, https://doi.org/10.5381/jot.2022.21.3.a7.
[19] L. De Moura, N. Bjørner, Z3: an efficient SMT solver, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2008, pp. 337–340.
[20] H. Wu, Finding achievable features and constraint conflicts for inconsistent metamodels, in: European Conference on Modelling Foundations and Applications, Springer, 2017, pp. 179–196.
[21] M.H. Liffiton, K.A. Sakallah, Algorithms for computing minimal unsatisfiable subsets of constraints, J. Autom. Reason. 40 (1) (2008) 1–33.