UMLtoCSP (UOST): A Tool for Efficient Verification of UML/OCL Class Diagrams Through Model Slicing

Asadullah Shaikh
The Maersk-McKinney Moller Institute University of Southern Denmark (Denmark) and Kulliyyah of Information and Communication Technology (KICT) International Islamic University (Malaysia)
ashaikh@mmmi.sdu.dk

Uffe Kock Wiil
The Maersk-McKinney Moller Institute University of Southern Denmark (Denmark)
ukwiil@mmmi.sdu.dk

ABSTRACT

Model errors are a major concern in the paradigm of Model-Driven Development (MDD) because of model transformations and code generation. It is important to detect model errors before transformation as in the later stages it is harder to trace and fix such errors. Formal verification tools and techniques can check the correctness of a model, but their high computational complexity can limit their scalability. In this research, we present a tool named UMLtoCSP (UOST) that uses a UML/OCL Slicing Technique (UOST) to verify complex UML/OCL class diagram. The tool accepts UML class diagrams annotated with OCL constraints as input, breaks the original model m into m_1, m_2, m_3,...,m_n sub-models while abstracting unnecessary model elements.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.4 [**Software Engineering**]: Software/Program Verification—*Model checking*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Verification Through Slicing, UML/OCL Model Verification, Complex Model Verification

1. INTRODUCTION

The technology of Model Driven Development (MDD) is growing day by day due the benefits of automatic model transformation and code generation. In MDD, Unified Modeling Language/Object Constraint Language (UML/OCL) class diagrams play an important role for model design, analysis, and transformation. Therefore, the verification of UML/OCL class diagrams at earlier stages in the development process is an essential task in order to check the correctness of model properties, i.e., verification of a UML/OCL class diagram with several OCL integrity constraints. "We have focused on static structure diagrams that describes the structure of a software system, UML class diagrams. Complex integrity constraints can be expressed in OCL. In this context, the fundamental correctness property of a model is *satisfiability*. Two different notions of satisfiability can be checked - either weak satisfiability or strong satisfiability. A class diagram is considered as weakly satisfiable if it is possible to create a legal instance/object of a class diagram which is non-empty, i.e., it contains at least one object from some class. Alternatively, strong satisfiability is a more restrictive condition requiring that the legal instance has at least one object from each class and a link from each association" [2, 4, 11–14].

In the current literature, there are few formal verification tools that check the correctness properties of models [3, 4, 8, 10], but typically these tools do not scale well. These tools can only verify UML/OCL class diagrams to a certain extent. For example, the tool UMLtoCSP [4] without our added slicing procedure is unable to verify a model having 15 classes, 10 association, 27 attributes, and 6 OCL invariants.

Our UMLtoCSP (UOST) tool [15] uses a different approach called model slicing to make verification of complex class diagram simpler. Afterwards, the tool uses bounded verification for automatic verification. Initially, the class diagram is partitioned into several independent submodels using a model slicing approach [12]. Secondly, all submodels are translated into Constraint Satisfaction Problem (CSP) [5]. Finally, the tool relies on the ECLiPSe constraint resolver [1] to compute whether the CSP has a solution or not. In case the CSP has a solution, the object diagram of each submodel will be generated with valid instances of the class diagram. If any submodel of an original class diagram cannot be verified, then the entire class diagram cannot be verified. Similarly, the entire class diagram will be satisfiable if all its submodels are satisfiable. The generation of objects proves the satisfiability.

The implemented approach in UMLtoCSP (UOST) is very fast; it can verify models within a few seconds. For example, a class diagram consisting of 15 classes, 10 associations, 27 attributes, and 6 OCL invariants could not be verified by UMLtoCSP within 2 hours. However, the UMLtoCSP (UOST) tool verified the same class diagram in 0.02 seconds. Our approach provides two main advantages over other similar verification tools. First of all, slicing is fully automated and can be implemented in any tool because it is neither tool

SIGSOFT'12/FSE-20, November 11–16, 2012, Cary, North Carolina, USA.

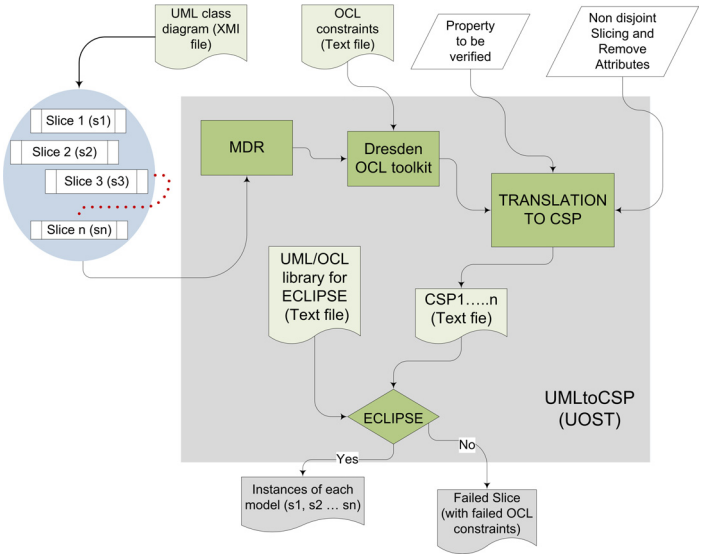


Figure 1: UMLtoCSP (UOST) execution flow.

dependent nor formalism dependent. Secondly, it points to the specific OCL constraint(s) that causes a UML/OCL class diagram to be unsatisfiable. For example, the technique detects unsatisfiable submodels and their integrity constraints among the complex hierarchy of an entire UML/OCL class diagram.

2. UMLTOCSP (UOST) FEATURES

The basic approach behind the UMLtoCSP (UOST) tool is a model slicing technique that enables efficient verification of UML/OCL class diagrams. The tool can verify different sets of properties for UML/OCL class diagrams with disjoint and non-disjoint sets of slicing. The features strong satisfiability and weak satisfiability are same as in UMLtoCSP [4]. However, other new features in UMLtoCSP (UOST) are:

Strong satisfiability: the class diagram should have a legal instance for at least one object from each class and a link from each association.

Weak satisfiability: the class diagram should have a legal instance/object which is non-empty, i.e., it contains at least one object from some class.

Remove attributes: for weak or strong satisfiability, unrestricted attributes can be removed from the class diagram.

Non-disjoint slicing: slicing of a class diagram with non-disjoint sets of submodels.

Disjoint slicing: slicing of a class diagram with disjoint sets of submodels.

Show specific invariants: detection of failing submodel(s) in disjoint slicing and a specific unsatisfiable invariant(s) in non disjoint slicing .

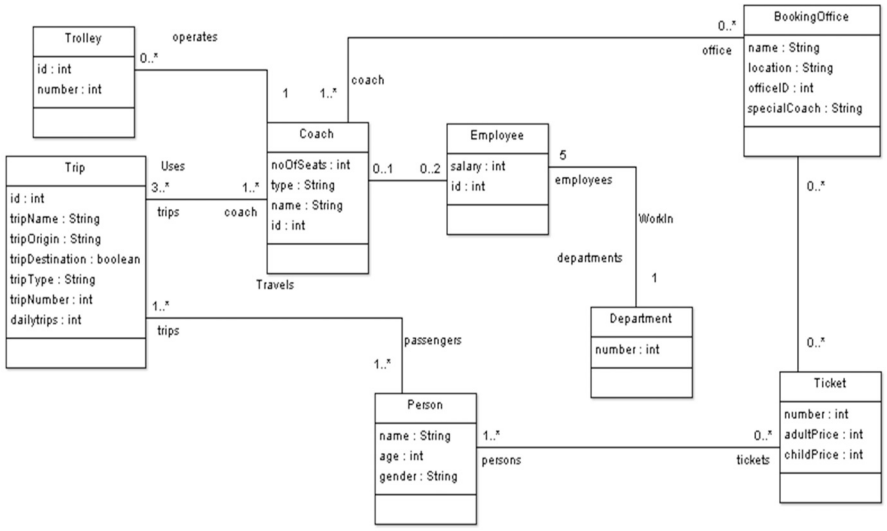
3. TOOL USAGE

In this section, we demonstrate the basic flow of the tool with the help of a running example. Figure 1 illustrates the execution flow of the tool. UMLtoCSP (UOST) uses several back-end services, for example, MDR [6] is used to parse the XMI files of the model slices. The Dresden OCL toolkit [7] is used to process the OCL constraints along with ECLiPSe solver in order to detect valid instances of a given input. Graphviz [9] is used to display the object diagram for all valid instances. In each verification tool, there is an *input*, a *process*, and an *output* (IPO). Therefore, we illustrate the working of the UMLtoCSP (UOST) through these general terms.

First of all, the user provides a UML/OCL class diagram as *input* in the form of an XMI file along with a text file specifying OCL constraints. Once the class diagram is loaded into memory, the user needs to provide a list of properties to be verified. The tool breaks the XMI file (class diagram) into several independent XMI files (slices of the input class diagram). After slicing, the user can set certain limits of the search space for a given input, i.e., a class diagram. This entire slicing process is automatic and invisible to the user. The next stage is the translation of the class diagram into a CSP, i.e., the *process*. Once the translation is done, the tool provides the *output* wether a valid instance of a given class diagram exists or not. If it does, the object will be generated for each slice. However, if it does not, the tool highlights the failing submodel with its corresponding OCL invariants. With the help of 'Show Specific Invariants', it is possible to highlight the specific failing constraints. The following section is comprised of the tool usage through a running example in order to provide a clear description of IPO.

3.1 Running example

Figure 2 shows the running example 'model Coach' that will be used to show the output of the tool. The 'model Coach' class diagram is annotated with 5 OCL invariants. First of all, the user needs to load the class diagram and OCL invariants into memory. Secondly, the approach identifies classes referenced by a given constraint. Such classes must be grouped and analyzed within the scope of the same slice. The process avoids repetition of same slices. UMLtoCSP (UOST) identifies OCL invariants and groups them if they restrict the same model elements. This is called *clustering of constraints* (Constraint Support). Thirdly, each cluster is passed to the eclipse solver for the translation into CSP. Figure 3 explores the translation process. However, before translating into CSP, it is important to select the property



context Coach **inv** MinCoachSize:
self.noOfSeats $\geq$ 10

context Trip **inv** IdUnique:
Trip::allInstances() −>isUnique (a | a.id)

context Person **inv** PersonSize:
Person::allInstances() −>size() = 1

context Person **inv** PersonsTicket:
Person::allInstances() −>isUnique (t | t.tickets.number)

context Department **inv** DeptartmentSize:
Department::allInstances() −>size() = 1

Figure 2: UML/OCL class diagram used as running example (model Coach).

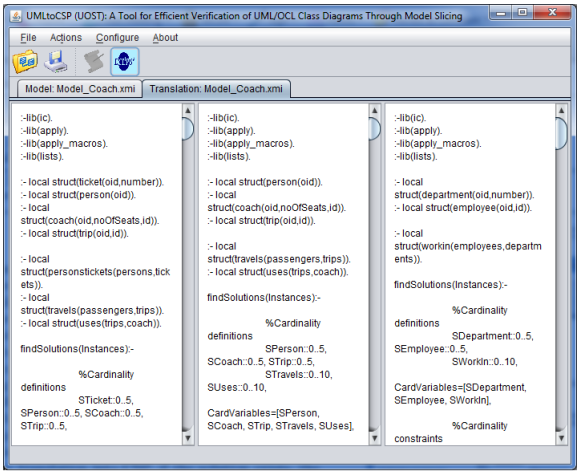


Figure 3: UMLtoCSP (UOST) translation process.

being verified. The next stage is the translation into CSP of each submodel. For 'model Coach' class diagram, in case of non-disjoint slicing, we will receive three submodels with strong satisfiability. However, in case of disjoint slicing two submodels will be received with strong satisfiability.

After the translation into CSP, if the solution exists, the tool will generate the respective object diagram for each submodel. For example, Figure 4 shows the desired output of the tool, i.e., the number of satisfiable submodels. Figure 5 shows the object diagram of slice 3. In the output screen, each submodel is shown with its corresponding invariants along with total submodel (slicing) time and total verification time.

If there is a complex class diagram (with hundreds of classes and invariants) having one of its properties failed, then UMLtoCSP (UOST) highlights the failed slice along with specific invariants. This new research approach detects unsatisfiable submodels and their integrity constraints among the complex hierarchy of an entire UML/OCL class diagram. The software engineers can therefore focus their revision efforts on the incorrect submodels while ignoring the rest of the model. The upper part of figure 6 highlights the unsatisfiable submodel (Submodel 1) along with its failed OCL constraints, i.e., two invariants, whereas, in the lower part the specific unsatisfiable constraint is shown.

4. CONCLUSIONS

This paper presents a research tool called UMLtoCSP (UOST) that can verify properties of UML class diagrams even when the class diagram is complex having hundreds of classes, attributes, associations, and invariants. The model slicing approach is implemented in the tool that helps to improve the efficiency of the verification process. Thanks to the slicing procedure UML/OCL class diagrams that were not verifiable before are now verifiable. The process of slicing and verification is fully automatic and it also provides constructive feedback if the class diagram is unsatisfiable. UMLtoCSP (UOST) detects the specific failed constraints and therefore, the designers only need to concentrate on

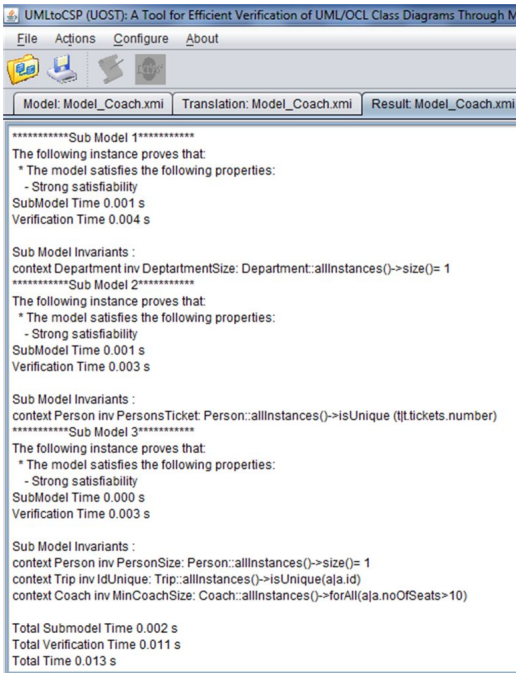


Figure 4: Slices with invariants.

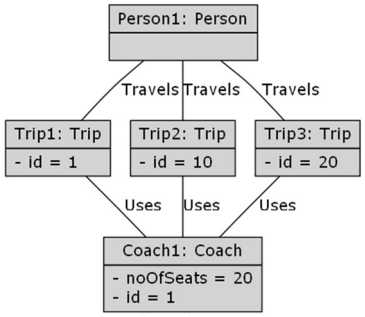


Figure 5: Object diagram of slice 3.

specific unsatisfiable properties while ignoring the rest of complex hierarchy.

5. REFERENCES

[1] K. R. Apt and M. G. Wallace. *Constraint Logic Programming using* ECL^{i}PS^{e}. Cambridge University Press, Cambridge, UK, 2007.

[2] M. Balaban and A. Maraee. A UML-based method for deciding finite satisfiability in Description Logics. In *DL'2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[3] A. Brucker and B. Wolff. Hol-ocl: A formal proof environment for uml /ocl. In *Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science, pages 97–100. 2008.

[4] J. Cabot, R. Clarisó, and D. Riera. UMLtoCSP: a tool for the formal verification of UML/OCL models using constraint programming. In *ASE'2007*, pages 547–548. ACM, 2007.

[5] J. Cabot, R. Clarisó, and D. Riera. Verification of uml/ocl class diagrams using constraint programming. In *Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pages 73–80. IEEE Computer Society, 2008.

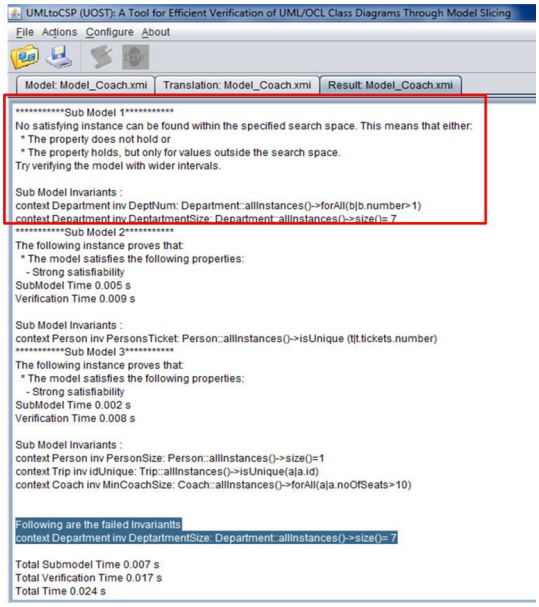


Figure 6: UMLtoCSP (UOST) output in case of unsatisfiable class diagram.

[6] J. Cabot, R. Clarisó, and D. Riera. Verification of uml/ocl class diagrams using constraint programming. In *ICSTW '08*, pages 73–80. IEEE Computer Society, 2008.

[7] B. Demuth. The Dresden OCL toolkit and its role in Information Systems development. In *ISD'2004*, Vilnius, Lithuania, 2004.

[8] M. Gogolla, J. Bohling, and M. Richters. Validation of uml and ocl models by automatic snapshot generation. In *In UML 2003*, pages 265–279. Springer, 2003.

[9] Graphviz. Graph visualization software. `http://www.graphviz.org`.

[10] D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, 2002.

[11] A. Queralt and E. Teniente. Reasoning on UML class diagrams with OCL constraints. In *ER'2006*, volume 4215 of *LNCS*, pages 497–512. Springer-Verlag, 2006.

[12] A. Shaikh, R. Clarisó, U. K. Wiil, and N. Memon. Verification-driven slicing of uml/ocl models. In *ASE*, pages 185–194, 2010.

[13] A. Shaikh, U. K. Wiil, and N. Memon. Evaluation of tools and slicing techniques for efficient verification of uml/ocl class diagrams. *Adv. Soft. Eng.*, 2011:5:1–5:18, 2011.

[14] A. Shaikh, U. K. Wiil, and N. Memon. Uost: Uml/ocl aggressive slicing technique for efficient verification of models. In *SAM 2010*, volume 6598 of *LNCS*, pages 173–192. Springer-Verlag Berlin Heidelberg, 2011.

[15] UMLtoCSP(UOST). A tool for efficient verification of uml/ocl class diagrams through model slicing. `http://asadshaikh.com/UMLtoCSP_UOST`.