UMLtoCSP: A Tool for the Formal Verification of UML/OCL Models Using Constraint Programming

Jordi Cabot
Estudis d'Informàtica, Multimèdia i Telecomunicació
Universitat Oberta de Catalunya
08018 Barcelona, Spain
jcabot@uoc.edu

Robert Clarisó
Estudis d'Informàtica, Multimèdia i Telecomunicació
Universitat Oberta de Catalunya
08018 Barcelona, Spain
rclariso@uoc.edu

Daniel Riera
Estudis d'Informàtica, Multimèdia i Telecomunicació
Universitat Oberta de Catalunya
08018 Barcelona, Spain
drierat@uoc.edu

ABSTRACT

We present UMLtoCSP, a tool for the formal verification of UML/OCL models. Given a UML class diagram annotated with OCL constraints, UMLtoCSP is able to automatically check several correctness properties, such as the strong and weak satisfiability of the model or the lack of redundant constraints. The tool uses Constraint Logic Programming as the underlying formalism and the constraint solver $\mathsf{ECL^iPS^e}$ as the verification engine.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.4 [**Software Engineering**]: Software/Program Verification; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search

General Terms

Verification

Keywords

UML, OCL, MDD, Model Verification, Constraint Programming

1. INTRODUCTION

The growing adoption of Model-Driven Development (MDD) methods, where models are the primary artifact of the development process, is shifting the focus of existing sofware engineering methods from code to models. In this context, the correctness of such models plays an important role in the quality of the final software system.

Unfortunately, there are currently few tools that support the verification of software models and, more precisely, the analysis of UML class diagrams annotated with OCL constraints (possibly the diagram most frequently used in automated code generation). The main reason is that the formal verification of a model is an undecidable problem. Therefore, current tools are either not fully automated or they work on a restricted subset of the UML and OCL languages.

Our UMLtoCSP tool [7] uses a different approach (*bounded verification*) to achieve an automatic and decidable verification. First, the initial model is translated into a Constraint Satisfaction Problem (CSP). More precisely, for each element of the class diagram we define a set of variables, domains and constraints that define a CSP. The correctness properties to be verified are translated as additional constraints of the CSP. Then, the tool relies on the constraint solver $\mathsf{ECL^iPS^e}$ [1] to determine whether the CSP has a solution or not.

A solution is an assignment of values from the domain to the set of variables in such that way that all constraints are satisfied. Each each solution defines a valid instance of the model. The existence of at least one solution proves that the model satisfies the correctness property.

This approach presents several benefits with respect to other related tools. First of all, it is fully automated, unlike theorem proving methods that may require user assistance (e.g. HOL-OCL [2]). The approach does not impose any underlying restriction on the expressiveness of the UML/OCL models. In this sense, it supports OCL constraints, unlike previous methods based on Constraint Programming [3]. Furthermore, it does not require the user to annotate, modify or enrich the original UML/OCL model in any way (unlike the validation tool USE [5], where the designer must write the ASSL code that creates valid snapshots). Finally, besides determining the correctness of the model, the tool is able to *explain* this result by means of computing and showing to the designer a valid instance that certifies it.

2. FUNCTIONALITY

The tool can verify several typical correctness properties on UML/OCL models:

Strong satisfiability: the model should have a finite instance where the population of each class and association is not empty.

Weak satisfiability: the model should have a finite instance where at least one class has a non-empty population.

Liveliness of a class X: The model should have a finite instance where the population of X is not empty.

Lack of constraint subsumption: Given two constraints c_1 and c_2, there should be a finite instance where c_1 is satisfied and c_2 is not satisfied. Otherwise, c_1 subsumes c_2 and therefore c_2 could be removed.

ASE'07, November 4–9, 2007, Atlanta, Georgia, USA.
ACM 978-1-59593-882-4/07/0011.

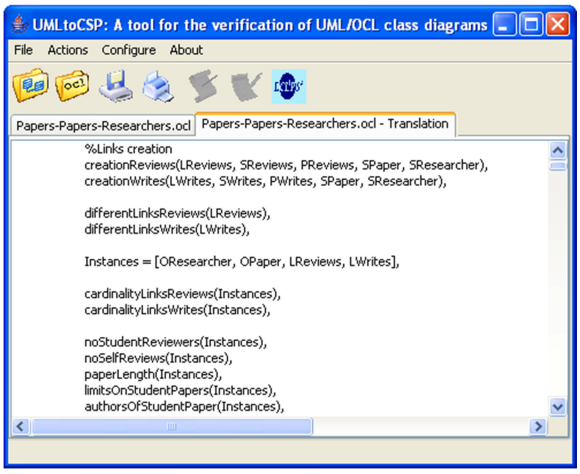

Figure 1: Graphical User Interface of UMLtoCSP.

Lack of constraint redundancy: Given two constraints c_1 and c_2, there should be a finite instance of the model that satisfies c_1 or c_2, but not both.

3. TOOL USAGE

UMLtoCSP works as a stand-alone application, either from the command line or from a graphical user interface (see Fig. 1). In both cases, the execution flow is depicted in Fig. 2: the user provides as an input an XMI file with the UML class diagram, a text file with the OCL constraints and the list of properties to be verified. Optionally, before starting the analysis, the user can tune the limits of the search space if he/she does not want to use the default values.

Once all these inputs have been provided, the translation of the model into a CSP and the CSP-based verification of the property are *fully automatic* and *transparent to the user*. The tool reports whether the property holds or not and, if it does, it depicts an instance of the model certifying it.

Internally, UMLtoCSP uses several existing libraries and tools: MDR is used to parse the XMI files, Dresden OCL toolkit [4] to process the OCL constraints, the constraint solver ECL^{i}PS^e to find the solutions of the CSP and the graph visualization package Graphviz [6] to display instances of the model. The tool itself is implemented in two components: a Java library (providing glue code, the GUI and the translation into CSP) and an ECL^{i}PS^e library written in Prolog (providing utility code for UML and OCL such as a Prolog implementation of the OCL Standard Library).

4. CONCLUSIONS

UMLtoCSP can verify quality properties of UML class diagrams with OCL constraints. The method is fully automatic and provides useful feedback to designers with a low overhead. As future work, we plan to extend the degree of OCL support (e.g. operations on strings), add support for other correctness properties (e.g. applicability of operations) and integrate UMLtoCSP as a plug-in inside other CASE tools.

5. ACKNOWLEDGMENTS

This work is partially funded by a research grant by the Internet Interdisciplany Institute (IN3) at UOC. The authors would like to thank Patricia de la Fuente and Christian Pérez-Llamas for their work on the implementation of the tool.

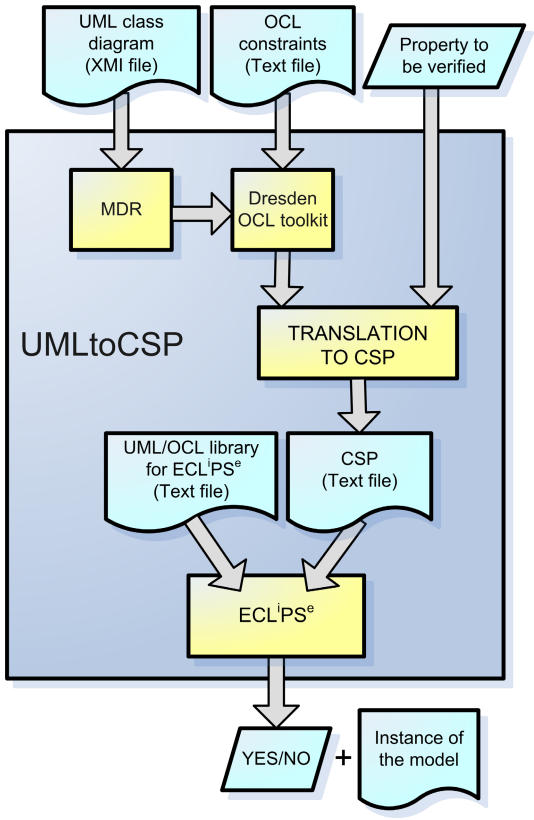

Figure 2: Architecture of UMLtoCSP.

6. REFERENCES

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

[2] A. D. Brucker and B. Wolff. The HOL-OCL book. Technical Report 525, ETH Zurich, 2006. .

[3] M. Cadoli, D. Calvanese, G. D. Giacomo, and T. Mancini. Finite satisfiability of UML class diagrams by Constraint Programming. In *Proc. of the 2004 International Workshop on Description Logics (DL'2004)*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[4] B. Demuth. The Dresden OCL toolkit and its role in Information Systems development. In *Proc. of the 13th International Conference on Information Systems Development (ISD'2004)*, Vilnius, Lithuania, 2004.

[5] M. Gogolla, J. Bohling, and M. Richters. Validating UML and OCL models in USE by automatic snapshot generation. *Journal on Software and System Modeling*, 4(4):386–398, 2005. .

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

[7] UMLtoCSP. A tool for the formal verification of UML/OCL models based on Constraint Programming. `http://gres.uoc.edu/UMLtoCSP`.