EMFtoCSP: A Tool for the Lightweight Verification of EMF Models

Carlos A. González*, Fabian Büttner*, Robert Clarisó†, and Jordi Cabot*
*École des Mines de Nantes - INRIA - LINA, Nantes, France
{carlos.gonzalez, fabian.buettner, jordi.cabot}@inria.fr
†Universitat Oberta de Catalunya, Barcelona, Spain
rclariso@uoc.edu

***Abstract*—The increasing popularity of MDE results in the creation of larger models and model transformations, hence converting the specification of MDE artefacts in an error-prone task. Therefore, mechanisms to ensure quality and absence of errors in models are needed to assure the reliability of the MDE-based development process. Formal methods have proven their worth in the verification of software and hardware systems. However, the adoption of formal methods as a valid alternative to ensure model correctness is compromised for the inner complexity of the problem. To circumvent this complexity, it is common to impose limitations such as reducing the type of constructs that can appear in the model, or turning the verification process from automatic into user assisted. Since we consider these limitations to be counterproductive for the adoption of formal methods, in this paper we present EMFtoCSP, a new tool for the fully automatic, decidable and expressive verification of EMF models that uses constraint logic programming as the underlying formalism.**

I. Introduction

Model-driven Engineering (MDE) is a popular approach to the development of software based on the use of models as primary artifacts. In an MDE-based software development process, the focus is on the design and creation of models to be (semi)automatically transformed into new models, and eventually into the code that will comprise the new software system, rather than on directly coding it by hand. The increasingly popularity of MDE has led to a growing complexity in models, thus turning their design and creation into an error-prone task, that may compromise the reliability of the development process and therefore the soundness of the resulting software system. For this reason, and to ensure the reliability of MDE-based processes, it is necessary the presence of mechanisms to ensure model correctness. Since model transformations can be seen as a particular kind of models [1], the approach to be followed at the time of verifying model transformations is the same followed to verify models.

Formal methods play an important role to ensure software correctness, but their adoption to ensure model correctness is compromised since, it is well known that formal verification of models is a very complex problem, undecidable in general, and especially in those cases in which models are extended with constraints expressed with the OCL. To overcome this problem, existing approaches typically limit model expressiveness by restricting the type of modeling and OCL constructs that may be used [2], [3], require user-interaction [4] or are semi-decidable [5]. However, in our opinion, these limitations compromise the adoption of formal methods as a valid alternative to ensure model correctness. Instead, we follow an approach based on bounded verification to ensure termination, automation and expressiveness of the verification process. A more detailed comparison with the other approaches can be found in the related work.

In this sense, this paper presents EMFtoCSP[1], an Eclipse[2] integrated tool for the verification of EMF[3] models annotated with OCL constraints. EMF is the de facto standard modeling framework in the industry. Unfortunately, there is no official Eclipse project devoted to the verification of EMF models. EMFtoCSP is a first step in this direction.

In EMFtoCSP, the initial model along with its constraints and the correctness properties to be checked, are translated into a constraint satisfaction problem (CSP). Then, a constraint solver is used to determine whether a solution for the CSP exists or not. The CSP is build such that the CSP has a solution iff the model plus the constraints satisfy the correctness property. If a solution is found, EMFtoCSP provides a valid instance of the input model to certify it.

As of now, EMFtoCSP supports the verification of the following correctness properties: strong satisfiability, weak satisfiability, lack of constraint subsumptions and lack of constraint redundancies. It is important to notice that there is a relationship between some of these properties, for example, strong satisfiability implies weak satisfiability and the lack of constraint subsumption between two constraints implies that none of them are redundant.

EMFtoCSP is as an evolution of a previous tool called UMLtoCSP [6] aimed specifically at the verification of UML class diagrams. Thanks to its more general scope, EMFtoCSP can be used to verify a larger variety of models but, noticeably, it can also be used to analyse the quality of domain-specific modeling languages by evaluating the

[1] http://code.google.com/a/eclipselabs.org/p/emftocsp/
[2] http://www.eclipse.org/
[3] http://www.eclipse.org/modeling/emf/

978-1-4673-1906-5/12/$31.00 © 2012 IEEE

correctness of their abstract syntax (e.g. by checking if it is possible to create models conforming to that metamodel).

This paper is organized as follows: Section 2 provides some background on constraint programming. Section 3 describes in detail how the CSP is built. Section 4 introduces EMTtoCSP, its usage and its overall architecture. Section 5 illustrates a performance analysis of the tool. Section 6 reviews the related work and, finally, Section 7 draws some conclusions and outlines some challenges we would like to address in a near future.

II. Constraint Programming

Constraint programming is an alternative approach to programming in which the programming process is limited to a generation of requirements (constraints) and a solution of these requirements by means of general or domain specific methods. In constraint programming, the constraints state the relations between a number of variables, and a constraint solver is employed to try to find a solution, that is, a value assignment to the variables that satisfies the constraints. Therefore, problems addressed by constraint programming are known as constraint satisfaction problems (CSP).

Formally speaking, a CSP is represented by the tuple $CSP = \langle V, D, C \rangle$ where V denotes the set of variables, D the set of domains, one for each variable, and C the set of constraints. Typically, constraints are described by a combination of arithmetic expressions, mathematical comparison operators and logical operators. As it was mentioned before, a solution to a CSP is an assignment of values to the variables that satisfies the constraints. In the case there is no solution, the CSP is called unfeasible.

At the time of looking for a solution, the most common technique used by constraint solvers is backtracking, combined with constraint propagation techniques. During this process, the solver attempts to assign values to variables following a certain order. If the partial solution violates any constraint, then the solver reconsiders the last assignment, trying a new value in the domain and backtracking to previous variables if there are no more values available. The process continues until a solution is found or all possible assignments have been considered. Termination is guaranteed by forcing all the variable domains to be finite. Constraint propagation techniques make the backtracking process much more efficient by pruning the search tree. To do that, information about the structure of the existing constraints and the decisions already made is employed to identify and avoid unfeasible values in the domain of unassigned variables.

EMFtoCSP utilizes a constraint solver called ECL^{i}PS^e Constraint Programming System[4]. This solver is capable of reasoning about, among others, boolean, interval, linear and arithmetic constraints. In ECL^{i}PS^e, constraints are expressed as predicates in a Prolog-based language, we will colloquially refer to as "ECL code" from now on, since CSPs must be stored in files with ".ecl" extension in order to be loaded into the solver.

[4]http://eclipseclp.org ; Do not confuse with the Eclipse IDE and Platform

III. CSP Generation

With our approach, the problem of verifying the correctness of a model is reduced to the problem if the CSP generated from it has a solution. We briefly show in this section how the translation process is performed, using the syntax provided by the ECL^{i}PS^e Constraint Programming System.

For the sake of efficiency, the verification problem can be split into two subproblems:

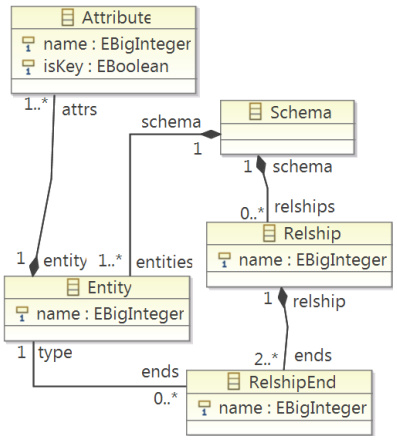
- **Subproblem 1**: choose a valid population size for the model, i.e. decide the number of instances of each class (objects) and association (links) that *may* provide a valid solution.
- **Subproblem 2**: given a specific population size, assign legal values to all attributes of objects and association ends of links and check if the assignment constitutes indeed a valid solution.

Both subproblems can be defined as CSPs and solved sequentially, using solutions to subproblem 1 as an input to subproblem 2. In the following, we characterize the contribution of each EMFtoCSP input element (models, constraints and properties) to each subproblem. Our running example will be the EMF model in Fig. 1, which describes a simple metamodel for Entity-Relationship diagrams annotated with several OCL invariants.

A. Model Translation

The model is the core of the CSPs for both subproblems, as it defines the relevant variables and domains:

- In subproblem 1, there is one integer variable for each class (e.g. `SizeSchema`, `SizeRelship`) and another for each association (e.g. `SizeAttributeEntity`). The domain of these variables goes from 0 to the maximum number of objects/links to be considered in the (bounded) search, a value which can be configured by the user of EMFtoCSP.
- In subproblem 2, the number of objects and links is fixed (by the previous subproblem). For each object, there is one integer variable (`oid`) and one variable per attribute (e.g. `name` and `isKey`). For each link, there is one variable per association end (e.g. `schema`, `entities`, ...). The domain of attributes is user supplied in the configuration of EMFtoCSP, while the domain of `oid` is set by the tool and directly related to the number of instances of each class. Finally, the domain of association ends is precisely the domain for `oid`s in the class adjacent to the association end.



(a)

context Schema **inv** ERN:
entities−>forAll(e: Entity | relships−>forAll(r: Relship |
e.name <> r.name))
context Schema **inv** RN:
relships−>forAll(r1, r2: Relship | r1.name = r2.name implies r1 = r2)
context Schema **inv** EN:
entities−>forAll(e1, e2: Entity | e1.name = e2.name implies e1 = e2)
context Entity **inv** EAN:
attrs−>forAll(a1, a2: Attribute | a1.name = a2.name implies a1 = a2)
context Entity **inv** KEY:
attrs−>exists(a: Attribute | a.isKey = true)
context Relship **inv** REN:
ends−>forAll(e1, e2: RelshipEnd | e1.name = e2.name implies e1 = e2)

(b)

Figure 1. Running example: (a) Metamodel for ER diagrams, (b) OCL invariants constraining the choice of identifier names.

Moreover, the graphical constraints in the model must be also captured in the CSPs:

- In subproblem 1, the multiplicity of association ends defines constraints over the population of the classes participating in the association. Also, inheritance hierarchies define constraints over the population of subclasses and superclasses, e.g. each instance of a class is also an instance of its superclasses.
- In subproblem 2, the multiplicity of association ends constrains the choice of values for the association end variables: there is a lower and upper bound to the number of times that an object may participate in an association. Inheritance hierarchies constrain the assignment of `oids`: an object should be given the same `oid` in a subclass as in the superclass, taking into account restrictions such as disjointness or completeness of the inheritance relationship. Finally, there are some additional well-formedness constraints that must be captured in the CSP, such as the uniqueness of `oids` within a class or the uniqueness of links in an association.

B. Constraints Translation

OCL invariants establish properties that must be satisfied by all objects of the context class. These properties are translated into constraints of subproblem 2 that refer to the variables of the CSP.

First, the OCL invariant is parsed as an *abstract syntax tree* (AST) where each node represents an expression: intermediate nodes are the operators and method calls and leaves are constants, attribute names, ... Each expression is translated into an ECL^{i}PS^e predicate `eval(Instances, Result)` that receives all variables of the CSP as a parameter and characterizes its result:

- **Leaf nodes** either set a constant value for the result or relate it to the value of a variable of the CSP. For example, the boolean constant `false` is translated into the predicate

```
evalConstantFalse( _, Result ) :-
  Result = 0.
```

  where "_" states that the result of this predicate does not depend on the rest of the variables of the CSP.
- **Intermediate nodes** describe the result as a combination of the result of its subexpressions. For instance, a node with an integer multiplication operator would be translated into the following predicate:

```
evalImplies( Instances, Result ) :-
  eval1stChild( Instances, Result1 ),
  eval2ndChild( Instances, Result2 ),
  =>(Result1, Result2, Result).
```

  This predicate does not compute the implication, as the variables of the CSP (`Instances`) do not have a value until a solution to the CSP is found. Instead, it states the relationship between the result of the implication and its subexpressions. This relationship will be used to guide the search process for a feasible solution, e.g. if `Result1` is false, then we know that `Result` is true without having to evaluate `Result2`.

Finally, the ECL^{i}PS^e predicate for the root node of the AST is evaluated on all objects of the context class, forcing its result to be true, i.e. `evalRoot(Instances, 1)`.

C. Properties Translation

Correctness properties state desirable conditions about models that we are interested in checking. Given a model M and a correctness property P, our goal is to compute a legal instantiation of M (one that satisfies all graphical and textual constraints of M) that is a *witness* of P, i.e. it proves that M satisfies P.

Currently, we consider two families of correctness properties: conditions about the *population* of the model, e.g. that it is not empty, or about the *relationship among constraints*,

Table I
CATALOG OF CORRECTNESS PROPERTIES

There is a legal instance of the model with a non-empty population for: • all classes and associations (**strong satisfiability**) • some class (**weak satisfiability**) Given a pair of OCL constraints of the model C_1 and C_2, it is possible to build an instantiation where: • C_1 holds and C_2 does not (**non-subsumption**, C_1 does not imply C_2) • one constraint holds, but not the other one (**non-redundancy**)

e.g. that no pair of invariants is equivalent. These conditions are encoded in the CSP as additional constraints in subproblem 1 (for conditions on the population size) or subproblem 2 (for conditions about the relationship among invariants).

Table I summarizes the correctness properties under consideration. Their translation into ECL^{i}PS^e constraints is straightforward. For example, weak satisfiability requires that the sum of all size variables in subproblem 1 is greater than zero:

```
weakSatisfiability( SizeVars ) :-
  sum(SizeVars) #> 0.
```

As another example, to check if constraints RN and EN from Fig. 1 (b) are non-redundant, a constraint is added to subproblem 2 stating that the root predicates of RN and EN evaluate to a different result:

```
nonRedundant_RN_EN( Instances ) :-
  evalRootRN( Instances, Result1 ),
  evalRootEN( Instances, Result2 ),
  Result1 #\= Result2.
```

IV. THE TOOL

Once the theoretical background has been introduced, we now describe the EMFtoCSP tool itself.

A. Usage

EMFtoCSP is integrated in the Eclipse IDE, so it is necessary to have Eclipse up and running in order to use EMFtoCSP features. Once the Eclipse environment is opened, launching EMFtoCSP consists simply in right-clicking on the model we want to verify and choosing "Validate model..." from the context menu[5] to display the EMFtoCSP GUI .

As can be seen in Fig. 2, EMFtoCSP provides a GUI in the form of an easy-to-use wizard that guides the user through a sequence of predefined steps to collect the user input for the verification process. Namely, selecting the file with the OCL constraints Fig. 2 (a), determining the limits of the search space Fig. 2 (b), choosing the properties to verify Fig. 2 (c), and finally establishing the location where the outputs will be located Fig. 2 (d).

[5]The launcher of EMFtoCSP can only be accessed from the "Package Explorer" view, so it is important to choose a perspective in which this view remains visible

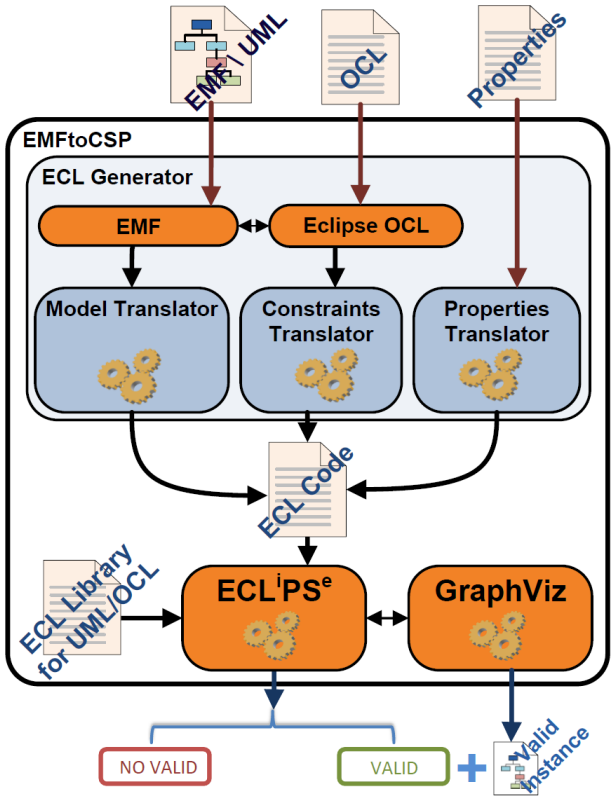


Figure 3. EMFtoCSP architecure

As the result, EMFtoCSP will display a message informing the user whether the input model satisfies the selected properties or not. If it does, EMFtoCSP will additionally output a valid instance of the input model that proves the property. An example of this can be seen in Fig. 2 (e), where a valid instance of the metamodel of Fig. 1 is displayed as a solution for the inputs provided in Figs. 2 (a), 2 (b) and 2 (c). Besides this, EMFtoCSP always provides the ECL code of the CSP generated as input for the CSP solver.

B. Architecture

The tool architecture can be seen in Fig. 3. User inputs are managed by the subsystem called "ECL Generator", which is in charge of generating the code to feed the solver with. In this subsystem, three different components are clearly distinguished each one coping with the different input elements that need to be translated, namely, the model, the set of constraints over the model and the properties to be checked. The EMF and OCL[6] parsers in Eclipse are used in the process.

Once the translation process has been performed, the generated ECL code is sent to ECL^{i}PS^e to check whether the input model holds the properties selected. Once ECL^{i}PS^e

[6]http://www.eclipse.org/modeling/mdt/?project=ocl

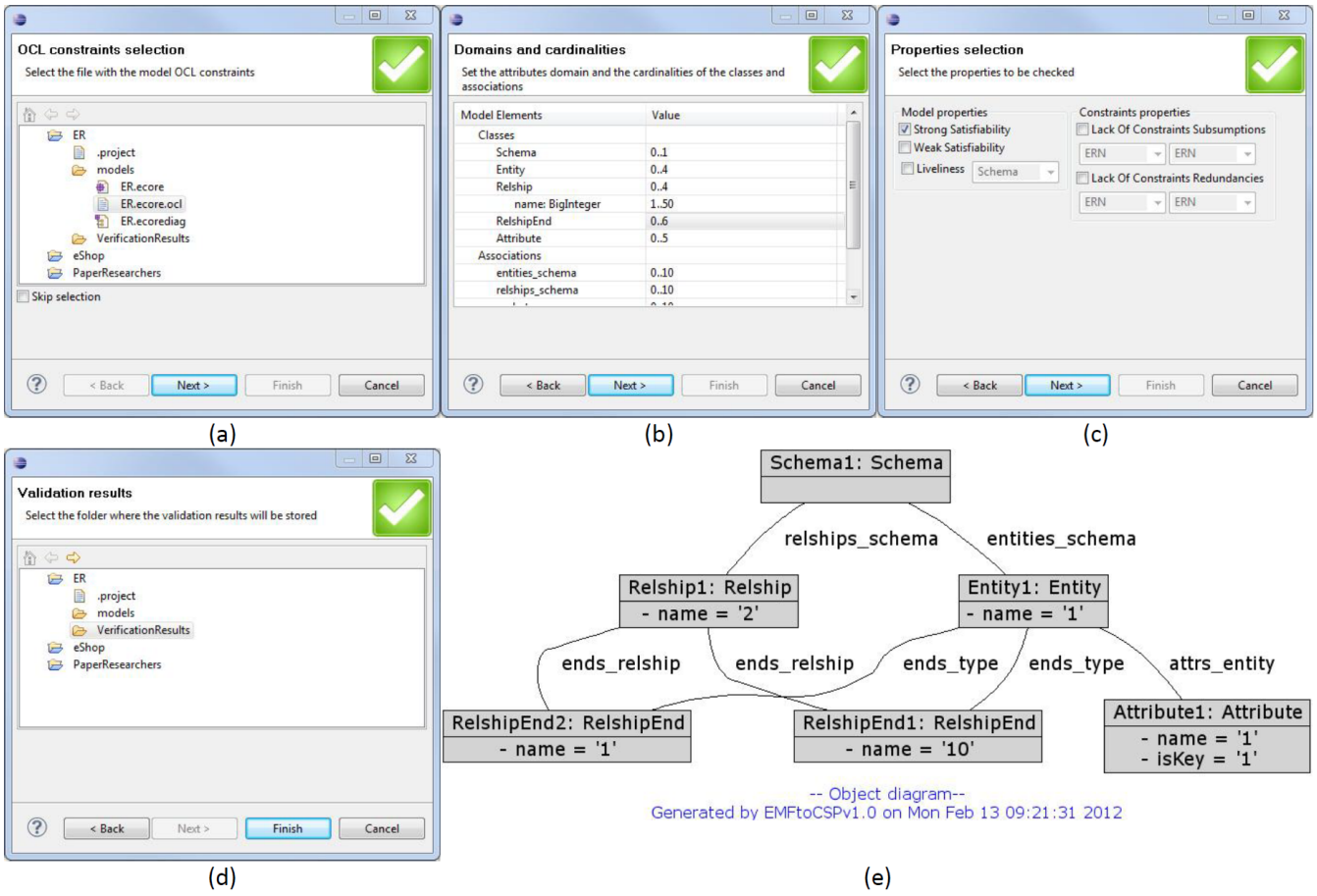


Figure 2. EMFtoCSP Graphical User Interface

finishes the search of a solution for the CSP, its feedback is interpreted and a message informing whether the input model holds the selected properties or not is displayed to the user. If the result is positive, the tool GraphViz[7] is used to graphically display the valid instance of the model identified as a solution by the solver.

It is worth noting that EMFtoCSP has been designed keeping in mind several possible extensions in the future. For instance, it is possible to plug modules translating models into other formalisms than the one used by ECL^{i}PS^e, as well as other solvers, provided that these modules respect the defined interfaces.

V. Performance

A tool like EMFtoCSP is only useful as long as it can scale when applied beyond toy examples. Though, of course, this is a work in progress, we have performed some experiments that show the applicability of the tool. The fact that the designer can decide herself the limits of the search space also facilitates using the tool with large models, where she can start by verifying the model using a small search space and expanding it later on if necessary.

In general, the scalability of the tool depends on the constraints of the model and the generated CSP. Two main factors are: (1) how much of the CSP can be solved using constraint propagation (and therefore avoiding backtracking), and (2) whether the CSP is non-trivially unsatisfiable (for the reason of symmetries).

This can be illustrated using the ER example from Fig. 1. Table II shows the runtimes of EMFtoCSP for several satisfiable ranges. All tests were conducted on a standard 2.2Ghz office laptop running Windows 7 and ECL^{i}PS^{e}6.0 with default settings. The ranges for RelshipEnd, the name attributes, and all link set sizes were set to 0..1000. We can see the tool finds ER instances of up to several hundred objects (in total) in reasonable time. For these cases, EMFtoCSP efficiently finds a valid link set (using linear constraint propagation to determine a valid link set size in the first step and using a global cardinality constraint as described in [7] in the second step). For this link set, a valid assignment of all attributes is then determined using linear constraint propagation, because at this time, the universal

[7]http://www.graphviz.org/

Table II
RUNTIMES FOR SAT CASES OF ER

Entities	Relships	Attributes	Runtime
1	1	1	$\leq$ 0.1s
10	10	50	0.76s
10	20	50	1.45s
20	20	50	1.99s
50	10	50	2.13s
50	20	50	3.30s
20	20	100	5.52s
50	20	100	9.71s
20	50	100	17.89s
50	50	100	24.91s

Table III
RUNTIMES FOR UNSAT CASES OF ER

Entities	Relships	Attributes	Names	Runtime
2	2	2	1	$\leq$ 0.1s
3	3	3	2	$\leq$ 0.1s
4	4	4	3	$\leq$ 0.1s
5	5	5	4	$\leq$ 0.1s
6	6	6	5	0.43s
7	7	7	6	5.77s
8	8	8	7	93.08s

quantifiers have been completely unrolled, leaving a purely linear problem. To make sure solutions found are non-trivial (e.g., all attributes connected to the first entity), we verified that the resulting runtimes are similar when changing the upper multiplicity bounds of the roles *entities*, *relships*, and *attributes* from "*" to 0..10.

Table III shows the runtimes for several "hard" unsatisfiable cases. In this setting, we restricted the range of the name attributes (i.e., the number of different names per type). There are not enough names to fulfill the corresponding constraints of the model. The table shows that EMFtoCSP scales much worse for these cases. The reason is that the (failing) attribute assignment is tried for all symmetrical link sets before reporting UNSAT. We hope to partly address this issue using a symmetry breaking during search approach such as described in [8].

While the employed example is very small in terms of the number of classes, we want to stress that our tool can also solve larger class diagrams, as the complexity of the search problem is not directly related to the number of classes in the model. On the contrary, given a number of objects (such as the 200 objects in the last line in table II), the search space is typically even smaller when these objects are distributed on more classes, because there are less possible combinations.

VI. RELATED WORK

To the best of our knowledge, there is an important lack of tools aimed at verification of EMF models in Eclipse. In fact, there is very limited tool support for the verification of static models under the presence of integrity constraints.

Most tools/approaches focus on the verification of UML/OCL class diagrams, based on the transformation of the diagram into a formalism where efficient solvers or theorem provers are available. However, since formal verification is a very complex problem, the selection of this formalism always entails some kind of trade-off. Some works avoid decidability problems by restricting to UML class diagrams without constraints (or just with some specific types of basic constraints). Among them, some are based on description logics [2], [9], [10] or constraint or linear programming [3], [11], [12]. Compared to them, our approach does not impose any theoretical restriction on the complexity of the OCL constraints that can be used, although at this moment, the current implementation of EMFtoCSP does not support yet all the features in the OCL 2.2 specification. Other tools are semidecidable (e.g. AuRUS [13], [5], based on query containment checking) or require user interaction to complete proofs like HOL-OCL [4] (based on higher order logics). However, both approaches provide complete proof procedures whereas EMFtoCSP, like all bounded verification methods, is decidable but not complete (absence of a solution in the search space cannot be used as a proof).

Other approaches follow our same bounded verification strategy. For instance, UML2Alloy [14] transforms UML/OCL classs diagrams into Alloy[8]. However, Alloy can not directly manipulate operations involving integers and works by transforming the problem into an instance of SAT (satisfiability of a boolean formula). The drawbacks of this are twofold. Firstly, numbers must be expressed in terms of boolean variables, specifying the number of bits used to encode each value, and secondly, operations on numbers must be encoded as boolean formulas operating at the bit level. All this can lead to a combinatorial explosion in the size of the formula when the bit-width of integers increases. In a CSP encoding arithmetic expressions is straightforward. Besides, UML2Alloy is not an integrated tool like it is the case with EMFtoCSP (where the access to the solver is transparent to the user). Another SAT-based solver with similar trade-offs but increased efficiency has recently been incorporated into the USE validation tool [15]. The approach of [16] translates the classes and constraints into first-order logic and employs SMT solvers to automatically reason about it. However, it is still open which class of models can be solved this way, and the translation tool is not available to the public. Finally, other approaches consider the use of search (genetic algorithms) to compute instances that satisfy a set of OCL invariants [17].

VII. CONCLUSIONS AND FUTURE WORK

In this paper we have presented EMFtoCSP, a tool for the fully automatic, decidable and expressive verification of EMF models extended with OCL constraints, based on their translation into a CSP such that the CSP has a solution iff the model satisfies the chosen correctness property.

Our approach follows a bounded verification strategy that ensures termination by limiting the search space when

[8]http://alloy.mit.edu/alloy/

looking for a solution for the CSP. Limits are created by restricting the number of instances per class and association and the domains of each attribute in the model. The trade-off is that the verification process is not complete (i.e. the CSP may have a solution beyond the considered search space), although we believe it is a reasonable trade-off considering the advantages of our method compared to alternative approaches. The translation of models into CSPs on infinitive domains is also possible, though. In these cases, constraint solvers allow incomplete search [18] where termination is not guaranteed and heuristics are needed to guide the search process. Under these circumstances our method would become semidecidable but complete (for properties that can be satisfied by finite instances).

In the future we plan to improve the efficiency of the generated CSP by refining our translation process. This can be done in a number of ways such as automatically defining appropriate ranges for attribute domains, based on an analysis of the OCL constraints that reference them, extracting implicit constraints to be used by the solver to improve constraint propagation [19], removing symmetries during the search phase [8] or partitioning the model to perform a parallel and independent (as much as possible) verification of each submodel, similar to [20].

EMFtoCSP is part of a more ambitious and collaborative vision for model verification. We plan to work on generalizing the tool infrastructure to transform it into a more global and extensible quality framework for EMF models. Our goal is to facilitate as much as possible that other research groups can integrate their verification approaches in this EMFQuality framework by focusing on their core expertise and relying on our infrastructure for all interface and model management aspects.

ACKNOWLEDGMENTS

This project has been partially funded by the Spanish Ministry of Science and Innovation through project "Design and construction of a Conceptual Modeling Assistant" (TIN2008-00444/TIN - Grupo Consolidado).

REFERENCES

[1] J. Bézivin, F. Büttner, M. Gogolla, F. Jouault, I. Kurtev, and A. Lindow, "Model transformations? transformation models!" in *MoDELS*, ser. LNCS, vol. 4199. Springer, 2006, pp. 440–453.

[2] D. Berardi, D. Calvanese, and G. D. Giacomo, "Reasoning on UML class diagrams," *Artif. Intell.*, vol. 168, no. 1-2, pp. 70–118, 2005.

[3] M. Cadoli, D. Calvanese, and T. Mancini, "Finite satisfiability of UML class diagrams by constraint programming," in *In Proc. of the 2004 Int. Workshop on Description Logics (DL2004), volume 104 of CEUR Workshop Proceedings. CEUR-WS.org*, 2004.

[4] A. D. Brucker and B. Wolff, "The HOL-OCL book," ETH Zurich, Tech. Rep. 525, 2006. [Online]. Available: http://www.brucker.ch/bibliography/abstract/brucker.ea-hol-ocl-book-2006

[5] A. Queralt and E. Teniente, "Verification and validation of UML conceptual schemas with OCL constraints," *ACM TOSEM*, vol. 21, p. to appear, 2012.

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

[7] J.-C. Régin, "Generalized arc consistency for global cardinality constraint," in *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1*, ser. AAAI'96. AAAI Press, 1996, pp. 209–215. [Online]. Available: http://dl.acm.org/citation.cfm?id=1892875.1892906

[8] I. Gent and B. Smith, "Symmetry breaking during search in constraint programming" *Proceedings ECAI*, vol. 2000, 2000.

[9] R. V. D. Straeten, T. Mens, J. Simmonds, and V. Jonckers, "Using description logic to maintain consistency between UML models," in *UML*, ser. LNCS, vol. 2863. Springer, 2003, pp. 326–340.

[10] M. Balaban and A. Maraee, "A UML-based method for deciding finite satisfiability in description logics," in *Description Logics*, ser. CEUR Workshop Proceedings, vol. 353. CEUR-WS.org, 2008.

[11] H. Malgouyres and G. Motet, "A uml model consistency verification approach based on meta-modeling formalization," in *SAC*. ACM, 2006, pp. 1804–1809.

[12] A. Maraee and M. Balaban, "Efficient reasoning about finite satisfiability of UML class diagrams with constrained generalization sets," in *ECMDA-FA*, ser. LNCS, vol. 4530. Springer, 2007, pp. 17–31.

[13] A. Queralt, G. Rull, E. Teniente, C. Farré, and T. Urpí, "AuRUS: Automated reasoning on uml/ocl schemas," in *ER*, ser. LNCS, vol. 6412. Springer, 2010, pp. 438–444.

[14] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "UML2Alloy: A challenging model transformation," in *MoDELS*, ser. LNCS, vol. 4735. Springer, 2007, pp. 436–450.

[15] M. Kuhlmann, L. Hamann, and M. Gogolla, "Extensive validation of OCL models by integrating SAT solving into USE," in *TOOLS (49)*, ser. LNCS, vol. 6705. Springer, 2011, pp. 290–306.

[16] M. Clavel, M. Egea, and M. G. de Dios, "Checking unsatisfiability for OCL constraints," in *OCL Workshop at MoDELS'09*, vol. 24. ECEASST, 2009.

[17] S. Ali, M. Z. Z. Iqbal, A. Arcuri, and L. C. Briand, "A search-based OCL constraint solver for model-based test data generation," in *QSIC*. IEEE Computer Society, 2011, pp. 41–50.

[18] K. R. Apt and M. Wallace, *Constraint Logic Programming using Eclipse*. Cambridge, UK: Cambridge University Press, 2007.

[19] F. Yu, T. Bultan, and E. Peterson, "Automated size analysis for OCL," in *ESEC/SIGSOFT FSE*. ACM, 2007, pp. 331–340.

[20] E. Uzuncaova and S. Khurshid, "Kato: A program slicing tool for declarative specifications," in *ICSE*. IEEE Computer Society, 2007, pp. 767–770.