ITP/OCL: A Rewriting-Based Validation Tool for UML+OCL Static Class Diagrams [⋆]

Manuel Clavel[1] and Marina Egea[2]

[1] Universidad Complutense de Madrid, Spain. clavel@sip.ucm.es
[2] Universidad Complutense de Madrid, Spain. marina_egea@fdi.ucm.es

Abstract. In this paper we present the ITP/OCL tool, a rewriting-based tool that supports automatic validation of UML class diagrams with respect to OCL constraints. Its implementation is directly based on the equational specification of UML+OCL class diagrams. It is written entirely in Maude making extensive use of its reflective capabilities. We also give notice of the Visual ITP/OCL, a Java graphical interface that can be used as a front-end for the ITP/OCL tool.

1 Introduction

The Unified Modeling Language (UML) [1] is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system. The UML notation is largely based on diagrams. However, for certain aspects of a model, diagrams often do not provide the level of conciseness and expressiveness that a textual language can offer. The Object Constraint Language (OCL) [2] is a textual constraint language. OCL comes to provide help on precise information specification in UML models.

Validation and testing in software development have been recognized of key importance for long. There are many different approaches to validation: simulation, rapid prototyping, etc. We *validate* a model by checking whether its instances (also called "snapshots") fulfill the desired constraints. A number of CASE tools exist which facilitate drawing and documenting UML diagrams. However, there is little support for validating models during the design stage and generally no substantial support for constraints written in OCL. In this paper we present the ITP/OCL tool, a rewriting-based tool that supports automatic validation of UML class diagrams with respect to OCL constraints. The ITP/OCL implementation is directly based on the equational specification of UML+OCL class diagrams developed in [3, 4]. It is written entirely in Maude [5], making extensive use of its reflective capabilities to implement the user interface, thanks to which the tool's underlying equational semantics remains hidden to the user, who only must be familiar with the standard notions of UML diagrams and OCL constraints.

[⋆] Research supported by Spanish MEC Projects TIC2003-01000, TIN2005-09207-C03-03, and by Comunidad de Madrid Program S-0505/TIC/0407.

2 UML+OCL Diagrams

The UML *static view* models concepts in the application domain as well as internal concepts invented as part of the implementation of an application. It does not describe the time-dependent behavior of the system, which is described in other views. Key elements in the static view are classes and their relationships, which can be of different kinds, including associations and generalizations. The static view is displayed in class diagrams.

Example 1. Consider the class diagram TRAINWAGON shown in Figure 1. It models an example from a railway context. A train may own wagons, and wagons may be connected to other wagons (their predecessor and successor wagons). Wagons can be either smoking or non-smoking.

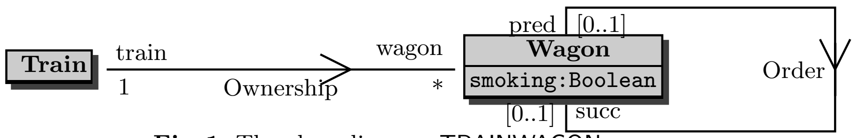

Fig. 1. The class diagram TRAINWAGON.

A system may be in different states as it changes over time. An *object diagram* models the objects and links that represent the state of a system at a particular moment. An *object* is an instance of a class. A *link* is an instance of an association. An object diagram is primarily a tool for research and testing.

Example 2. Consider now the object diagram TRAINWAGON-1 shown in Figure 2. It describes a snapshot of the railway system modeled by the class diagram TRAINWAGON, although possibly an “undesired” one since it describes a train with two wagons linked in a cyclic way!

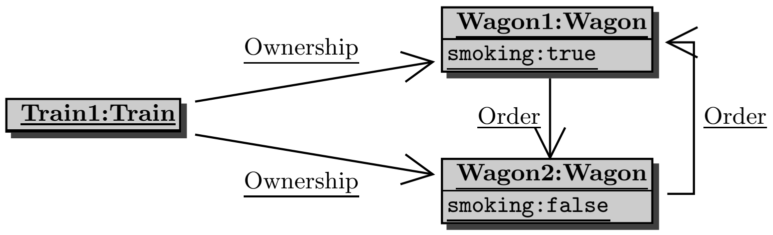

Fig. 2. The object diagram TRAINWAGON-1.

OCL is a pure specification language on top of UML. It is a textual language with a notational style similar to common object oriented languages.

Example 3. Consider the following constraint over the class diagram TRAINWAGON: *There do not exist two different wagons directly linked to each other* *in a cyclic way.* This constraint can be expressed using OCL as the following invariant notInCyclicWay over the class diagram TRAINWAGON:

```
not ((Wagon.allInstances)
    →exists(w1:Wagon | (Wagon.allInstances) →exists(w2:Wagon |
          w1:Wagon <> w2:Wagon
          and (w1:Wagon.succc) →includes(w2:Wagon)
          and (w2:Wagon.succ) →includes(w1:Wagon)))) .
```

The object diagram TRAINWAGON-1 does not satisfy this constraint, since there exists two wagons, namely Wagon1 and Wagon2, such that the successors of Wagon1 include Wagon2, and the successors of Wagon2 include Wagon1.

3 The ITP/OCL Tool

The ITP/OCL tool is based on the equational specification of UML+OCL class diagrams developed in [3, 4], according to which: i) class and object diagrams are specified as membership equational theories; ii) constraints are represented as Boolean terms over extensions of those theories; and iii) validating object diagrams with respect to constraints is reduced to checking whether the corresponding Boolean terms rewrite to true or false. The ITP/OCL tool is written entirely in Maude [5], a term-rewriting based programming language that implements membership equational logic (and rewriting logic). Maude is also a reflective programming language. This means, in particular, that both its parser and its rewriting engine are available to the programmer as built-in operations: we have taken advantage of the latter to implement the tool's OCL parser and of the former to implement the tool's UML+OCL rewriting-based validation engine. The implementation of the ITP/OCL tool comprises around 4,000 lines of Maude code. The latest version of the ITP/OCL tool, with the available documentation and a collection of examples (that includes class diagrams with enumeration classes, association classes, generalizations, attributes and query operations), can be found at `http://maude.sip.ucm.es/itp/ocl/`.

The implementation of an interactive tool in Maude comprises four different tasks: defining a read-eval-print loop; defining the syntax for the commands; defining the interaction with the loop; and defining the processing of the commands. Maude provides a generic input/output facility through its "loop objects" terms. It also provides great flexibility to define the syntax for the commands thanks to its mixfix front-end and to the use of *bubbles* (any nonempty list of Maude identifiers). Finally, the processing of the requests made to an interactive tool is defined in Maude by equations acting on the loop objects terms.

The ITP/OCL's commands can be grouped in four classes:

– *Commands that create a diagram.* They are defined by equations that add to the tool's database the module that, according with the tool's semantics, specifies an empty class (respectively, object) diagram. For example, to create a class diagram we use the command `(create-class-diagram` *CD* `.)`, where *CD* is the class diagram's name.

– *Commands that insert an element (class, attribute, association, and so on) in a diagram.* They are defined by equations that add to the module specifying the diagram in the tool's database the declarations (sorts, operators, memberships, equations) that, according with the tool's semantics, specify that the diagram has this element. E.g., to insert a class we use the command `(insert-class` *CD* `:` *C* `.)`, where *CD* is the class diagram's name and *C* is the class' name.
– *Commands that state a constraint over a class diagram.* They are defined by equations that associate to the module specifying the class diagram in the tool's database the Boolean term that, according with the tool's semantics, represents this constraint. For instance, to state a *contextualized invariant* we use the command `(insert-invariant` *CD* `:` *C* `:` *INV* `.)`, where *CD* is the class diagram's name, *C* is the contextual class' name, and *INV* is the invariant.
– *Commands that validate an object diagram.* They are defined by equations that check whether the Boolean terms representing the invariants reduce to true or false in the module that, according with the tool's semantics specifies the union of the class diagram, along with its invariants, and the object diagram. E.g., to check whether an object diagram validates the invariants stated over the class diagram of which it is an instance, we use the command `(check-invariants` *CD* `:` *OD* `.)`, where *CD* is the class diagram's name and *OD* is the object diagram's name.

4 The Visual ITP/OCL Tool

The Visual ITP/OCL is a Java graphical interface for the ITP/OCL tool.[3] Events on the Visual ITP/OCL's worksheets and toolbars are transformed into ITP/OCL commands and are interpreted and executed in a Maude process running the ITP/OCL tool. In its current state:

– Diagrams can be graphically created in a similar way to other CASE tools, like Rational Rose [6] or Gentleware Poseidon UML [7]. By clicking, dragging, and dropping on the class diagram's (respectively, object diagram's) worksheet and toolbar, the user can interactively insert, modify, and delete classes and associations (respectively, objects and links.) Several class diagrams can be opened at the same time. Object diagrams are associated to their class diagrams. There is also a zooming facility to increase/decrease the scale. Finally, diagrams can also be saved (and loaded from) a MySQL database and can be exported as eps-files.
– Constraints on class diagrams (respectively, queries on object diagrams) are inserted through an OCL editor and parser. Constraints inserted in a class diagram can be automatically checked over specific object diagrams. Queries can also be automatically evaluated.

[3] The Visual ITP/OCL tool is being developed by F. Alcaraz, J. P. Gavela, and J. Arias as a Master's project.

5 Related and Future Work

A number of CASE tools exists which facilitate drawing and documenting UML diagrams [6–8]. However, there is little support for validating models during the design stage and generally no substantial support for constraints written in OCL. The USE tool [9] is, however, a significant exception. The USE tool expects as input a textual description of a model and its constraints. This textual description is then displayed in a graphical interface. Objects and links can be then graphically created by a drag and drop facility. In every system state, the constraints can be automatically checked. The USE tool also supports the validation of sequence diagrams by checking that, for each step in the sequence, the initial and the resulting diagrams satisfy, respectively, the pre- and post-conditions constraining the application of the corresponding (non-query) operation. This feature is not yet supported by the ITP/OCL tool. Notice, however, that validating sequence diagrams à la USE only requires the capability of checking constraints (the pre- and post-conditions) over object diagrams. We plan to add this feature in the next version of the ITP/OCL tool.

The ITP/OCL tool is based on the equational specification of UML+OCL class diagrams: validating invariants over (or evaluating OCL queries in) object diagrams is done by rewriting the corresponding term in the corresponding equational specification. This is clearly different from the USE tool's underlying semantics and its corresponding evaluation mechanism [10]. Finally, as suggested by various of our referees, we have tried a first comparison between both tools. In particular, Table 1 shows the time (in seconds) consumed the tools to validate a number of constraints over two object diagrams, TRAINWAGON-10x25 and TRAINWAGON-10x100, of the class diagram TRAINWAGON. In addition to the constraint notInCyclicWay introduced in Section 2, we have considered the following constraints:

– *All trains must own at least one wagon.*

```
context Train inv atLeastOnewagon:
        self:Train.wagon size() ≥ 1.
```

– *A wagon and its successor wagon should belong to the same train.*

```
context Wagon inv belongToTheSameTrain:
        self:Wagon.succ →notEmpty() implies
        self:Wagon.succ →forAll(w:Wagon | (w:Wagon.train = self:Wagon.train)).
```

– *All trains will have the same number of wagons.*

```
context Train inv sameNumberOfWagons:
        Train.allInstances →forAll (t1:Train |
        (self:Train <> t1:Train implies
        (self:Train.wagon →size() = t1:Train.wagon →size()))).
```

The object diagram TRAINWAGON-10x25 contains 10 trains and 250 wagons, each train is linked to 25 different wagons, which are linked in the expected way; that is, each wagon has a predecessor and a successor, except for the first and the last wagon. The object diagram TRAINWAGON-10x100 contains 10 trains and 1000 wagons, each train is linked to 100 different wagons, which are also linked in the expected way. The object diagrams TRAINWAGON-10x25 and TRAINWAGON-10x100 satisfy indeed our four constraints. The validations have been carried out in a laptop computer with a 2GHz Pentium processor and 1 GB RAM. As expected, validating the constraint notInCyclicWay takes more time; essentially, it has to make, respectively, 250×250 and 1000×1000 comparisons. However, the time consumed by the USE tool is unexpectedly high. We have tried to obtain from the USE community an explanation for this fact (which may be simply due to our inexpert use of the tool) but we have not get an answer yet; as soon as we get it, we will publish it in the ITP/OCL web page.

Table 1. Validation times

	TRAINWAGON-10x25		TRAINWAGON-10x100	
	USE	ITP/OCL	USE	ITP/OCL
atLeastOnewagon	0.055s	0.076s	0.034s	0.116s
belongToTheSameTrain	0.207s	0.076s	0.970s	0.780s
sameNumberOfWagons	0.202s	0.120s	0.241s	0.936s
notInCyclicWay	45.745s	13.788s	2819.410s	233.710s

References

1. Object Management Group: Unified Modeling Language Specification (2004) `http://www.uml.org`.
2. Object Management Group: Object Constraint Language Specification (2004) `http://www.omg.org`.
3. Egea, M.: ITP/OCL: a theorem prover-based tool for UML+OCL class diagrams. Master's thesis, Facultad de Informática, Universidad Complutense de Madrid (2005) `http://maude.sip.ucm.es/~marina/`.
4. Clavel, M., Egea, M.: Equational specifications of UML+OCL static class diagrams. `http://maude.sip.ucm.es/itp/~clavel` (2006)
5. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude Manual (Version 2.2). (2005) SRI International, December 2005, `http://maude.cs.uiuc.edu`.
6. IBM: Rational Software (2006) `http://www-306.ibm.com/software/rational/`.
7. AG, G.: Poseidon Standard Edition (2006) `http://www.gentleware.com`.
8. Demuth, B., Löcher, S., Zschaler, S.: Structure of the Dresden OCL toolkit. Technical report, Technical University of Darmstadt, Germany (2004) Reviewed Conference Paper.
9. Richters, M.: The USE tool : A UML-based specification environment. (2001) `http://www.db.informatik.uni-bremen.de/projects/USE/`.
10. Gogolla, M., Richters, M., Bohling, J.: Tool support for validating UML and OCL models through automating snapshot generation. In: Proceedings of SAICSIT. (2003) 111–120