Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 102 (2004) 99–110
www.elsevier.com/locate/entcs

Ensuring UML Models Consistency Using the OCL Environment

Dan Chiorean Mihai Paşca Adrian Cârcu Cristian Botiza
Sorin Moldovan

Babes-Bolyai University, Computer Science Research Laboratory
str. M. Kogalniceanu, 1
400084 Cluj-Napoca, Romania

Abstract

The topic of UML model consistency is becoming increasingly important. Having a tool that checks the consistency of UML models is very useful. Using the XMI standard, the consistent models can be transferred from the checker tool to any other UML tool. By means of practical examples, this paper shows that using a framework based on OCL is a valuable approach when checking UML models. The results obtained in the examples highlight some shortcomings in the UML definition and prove that OCL offers the support needed in managing tool peculiarities.

Keywords: OCL, UML, XMI, metamodel, model consistency checking

1 Introduction

Checking UML model consistency is becoming a more stringent problem every day. First, the number of UML users is continuously increasing and thus the number of models constructed and transferred between different users and tools is increasing, too. Furthermore, applying MDA in the process of application development implies the transformation of models being correct with respect to the semantics of the modeling language. Detecting and removing model errors in the early stages of software development leads to shorter time-to-market and lower costs. Generally, model construction takes place in the

[1] E-mail: mailto:chiorean@cs.ubbcluj.ro

1571-0661 © 2004 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2003.09.005

framework of a methodology; moreover, each model is built for a certain application domain and is eventually implemented in a programming language. Therefore, checking the model against a set of methodological rules, application profile dependent rules (web applications, component-based applications, and so on), target programming language rules are important operations whose automation is possible and useful. OCL allows for a simple and suggestive specification of all the categories of rules mentioned above. OCL has been a part of UML since the first publication of the UML standard, being the only textual formalism used for the definition of both the UML static semantics and the user model semantics. As specified in [6], OCL is used to complement the information in UML diagrams.

When creating new entities, existing UML tools perform some checks, forbidding: name clashes (in case of some model elements), circular generalization relationships, transitions to input states or transitions from output states, and so on. These checks do not ensure the consistency of UML models as defined by means of Well Formedness Rules [2]. Consequently other checks have to be performed, at least at the time of model transition – from analysis to design or from design to implementation.

2 State of the art

Each UML tool supports different kinds of checks using different approaches. The most flexible strategy is by far the one adopted by tools supporting scripting languages. Rational Rose [3], Together [15], Objecteering [14] are among the best tools included in this category. But even this approach has some drawbacks:

- Many CASE tool repositories do not fully comply with the UML standard (all the above-mentioned tools are in this situation).
- The APIs of the repositories do not provide all the necessary information (for instance, most of the Additional Operations [3] are not available).
- The WFR specified in OCL have to be translated into scripts.

Moreover, the errors identified using any of these approaches are only reported. The user is not given the possibility to navigate the model to the sources of these errors, even if some corrections are suggested.

[2] The Well Formedness Rules, shortly referred to as WFR, are invariants on the UML metamodel classes, defining the UML static semantics.

[3] The Additional Operations, shortly AO, are observers defined on the metamodel classes to facilitate a simpler and expressive specification of the invariants on these classes.

As a result, by adopting the above-mentioned approaches, only a part of the checks needed to guarantee the consistency of UML models can be realized.

3 Object Constraint Language Environment

The Object Constraint Language Environment, shortly OCLE, is a UML CASE tool offering full OCL support, both at the metamodel and the model level. This tool was conceived, designed and implemented by the UBB LCI team see [16].

In this section we will present the main OCLE features needed in understanding the examples discussed in this paper. OCLE currently supports UML 1.5 and OCL 2.0. The tool is XMI 1.0, 1.1 and 1.2 compatible. Therefore it can load UML models produced by most of the CASE tools currently available (Together, Rational Rose, MagicDraw, Poseidon). The tool also exports UML models in XMI format, so that they can be later imported and modified in any other tool that supports XMI. After the UML models are created or loaded, they can be edited using a powerful property sheet, a customizable model browser and a UML diagram editor. The architecture of the tool is component-oriented, separating the concepts involved with respect to the Model-View-Controller pattern (MVC). Due to this architectural decision, the users can easily navigate among different views of the same information. The browsers and the diagram editors support information filtering, helping the users to represent only the information required in different situations. The tool offers a search capability at both the repository level and the OCL specifications level.

The most important feature of OCLE is the ability to perform different kinds of checks on the UML models and to correct the identified errors. This feature will be described in detail in the following paragraphs.

The checking process is based on the OCL formalism. This decision was influenced by the following factors:

- OCL is part of the UML standard, being used irrespective of the modeling level (model, metamodel, meta metamodel).
- The use of OCL reduces the number of formalisms required in producing and checking UML models, thus enlarging the number of potential users.
- The language is suggestive, easy to understand and to use.
- The WFR are specified in OCL [6], so these rules do not need a translation into another formalism.
- The constraint language is extensible and supports reusability at the metamodel level.

The OCL features of the tool comply with OCL 2.0 including tuples and nested collections. To offer a better management of OCL constraints, OCLE stores them in text files. An OCL text file can only contain specifications concerning a single modeling level (metamodel and user model levels are currently supported). The tool assists the user in defining OCLE projects. An OCLE project comprises one or more UML models (a single model being active at a given moment), one or more metamodel constraint files and (or) one or more model constraint files (see Figure 1).

Working with OCLE is thus simple and natural. The support offered by OCLE in identifying inconsistencies and eliminating them is very strong and flexible, as we will see in the following section.

The process of checking the consistency of a UML model with regards to a set of rules is divided into two phases. The first phase is the compilation of the OCL rules. This phase must complete successfully in order to allow the start of the second phase. The second phase assumes the evaluation of the rules, and OCLE provides three flexible ways of performing it: full batch, partial batch and single evaluation. The full and partial batch evaluations are an automated way of evaluating a set of OCL specifications over a set of selected model elements that match the contexts of these rules. Single evaluation is a finer-grained operation: it allows the evaluation of a single OCL expression (or even a part of it, as long as it makes sense) for a model element specified by the user. To perform a full or a partial batch evaluation, the user must first create (or open) an OCLE project, comprising the elements mentioned above. The compilation phase iterates over the OCL specification files included in the active project and analyzes their syntactic and semantic correctness. All these files are still seen as a whole, so that a definition written in one file (using the "def-let" mechanism) may be used in another file as long as both files are included in the same OCLE project. In this way, OCLE offers a valuable support for the reusability of the specifications expressed at the metamodel level.

Once the batch evaluation phase is completed, OCLE presents a report of the evaluations requested, performed and failed and also the detailed information regarding the failures. This information is grouped by metaclass, rule and model element and includes links that permit the quick identification of the OCL rules and UML model elements involved in each failure (see Figure 4). With its "single evaluation" feature, OCLE assists the user in discovering the possible source(s) of a problem by allowing the evaluation of sub-expressions. Using this feature, the user can trace the partial results of a certain expression for a model element set as context, simulating a step-by-step execution.

After the cause of the failure was detected, the tool assists the user in

correcting the model by creating new elements, deleting existing elements or by changing the values of element properties. The corrected model can then be saved in one or more XMI files and transferred to other tools.

4 Two examples of checking UML model consistency

4.1 The "ordersys" model

The first model chosen to exemplify the checking of UML model consistency against the WFR is "ordersys". The Order System sample is an order system application developed for a seafood distributing company. This is a real model including different Visual Basic libraries.

Delivered by Rational in the "samples" package of Rational Rose, the example contains the application model and the associated Visual Basic project. The model was exported in XMI 1.1 format using the Unisys Rose add-in and imported in OCLE. The OCLE project contains the UML model and the set of OCL constraints stored in several files (see Figure 1).

The rules used in this example are the WFR defining the UML static semantics. All the Additional Operations were grouped in one file. The WFR expressed by means of class invariants were stored in six other files. This grouping enables both a better management of the specifications and a finer reusability at the specification files level.

This UML model represents a late design model, including the libraries of the target programming language, all grouped in the `COM` package, as shown in the "UserModel" browser in Figure 1.

Considering that the associated Visual Basic project is executable, we are interested in seeing whether the design model information is consistent with the implementation model information. Maintaining consistency between these two models is an important task for developers.

The result of the first full batch evaluation is displayed in the log panel (Figure): 24 problems were found out of 22671 evaluations performed. However, despite the large number of evaluation operations requested, the whole operation took around 10 seconds on an Intel Pentium III system (1GHz with 512 MB of RAM), running Windows 2000.

In the following paragraphs we will look at the causes of some of the reported errors. All the rules whose evaluation failed for at least one model element were posted in the "Evaluation" pane. The first rule we consider is defined in the context of the `Namespace` metaclass:

```
context Namespace

inv WFR_1_Namespace:
```

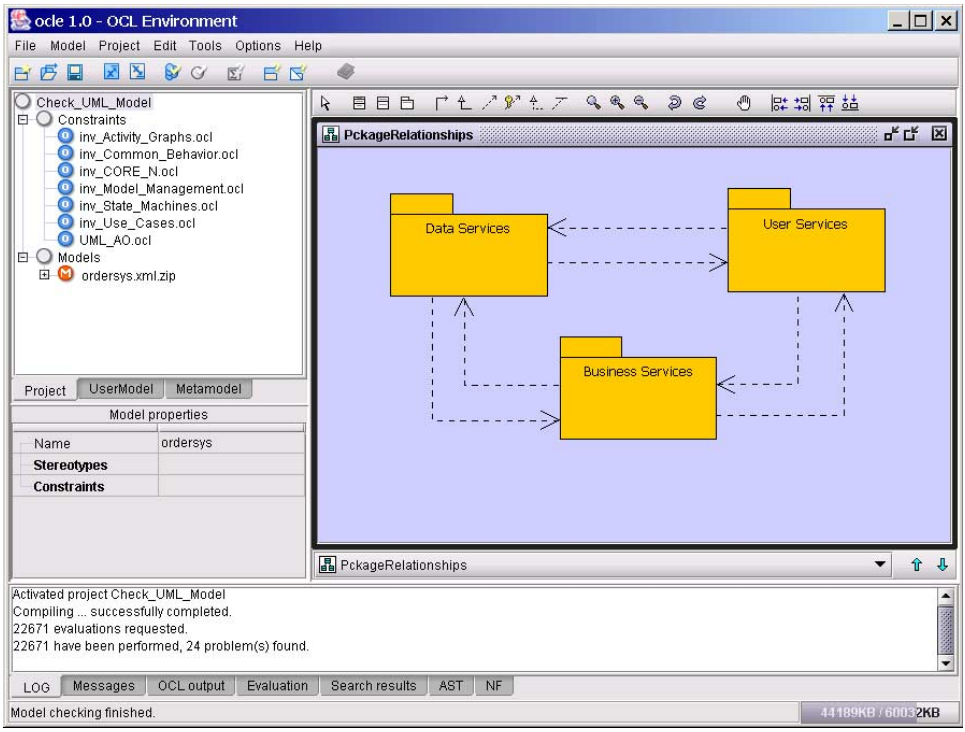

Fig. 1. The Project browser, the log output pane and a class diagram

```
-- [1] If a contained element, which is not an Association or Generalization
-- has a name, then the name must be unique in the Namespace.

    let noe:  Set(ModelElement) = self.ownedElement->reject(e |
        e.oclIsKindOf(Association) or e.oclIsKindOf(Generalization)) in
    if noe->reject(e | e.name='')->isUnique(e | e.name)
        then true
        else noe->select(e | noe->exists(ae | ae <> e and e.name =
            ae.name))->sortedBy (e | e.name)->isEmpty
    endif
```

A direct translation of the informal specification is simpler:

```
self.ownedElement->reject(e | e.oclIsKindOf(Association) or
e.oclIsKindOf(Generalization))->isUnique(e | e.name)
```

This direct specification was replaced with the one mentioned above for practical reasons. The unnamed model elements were rejected because, apart from associations and generalizations, there are other model elements (like dependencies, instantiations, etc.) whose names cannot be set in most of the current CASE tools. Consequently, all model elements that are not referred by name have to be rejected. The ‘‘`else`’’ branch of the ‘‘`if`’’ instruction was included exclusively to provide the user only with useful information in this

situation (the list of model elements having the same name). In our example, the elements causing the failure of the rule are four `ClassifierRoles`, two of them being named `NewRow`, and the others being named `dlg_Order` (see the OCL output pane and the "UserModel" browser in Figure 3). Changing the names of two of these model elements will solve the problem. This operation can be realized using the property sheet.

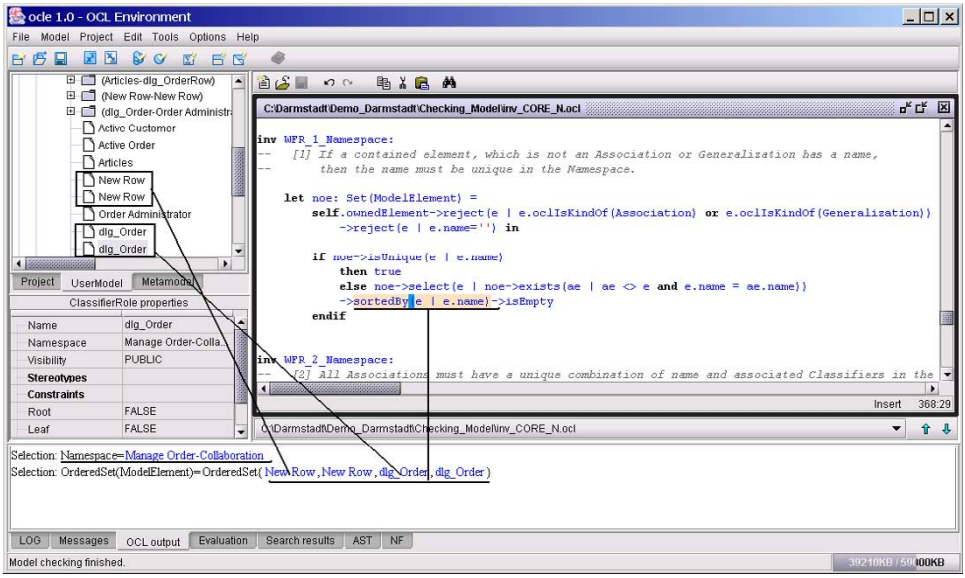

Fig. 2. Two name conflicts identified in the Manage Order-Collaboration Namespace

The second rule evaluated to `false` is:

```
context StructuralFeature
inv WFR_2_StructuralFeature:
-- [2] The type of a StructuralFeature must be a Class, DataType or Interface.

    if self.owner.stereotype.name->includes('enumeration')
        then true
        else self.type.oclIsKindOf(Class) or self.type.oclIsKindOf(Interface)
            or  if self.type.oclIsKindOf(DataType)
                    then (Set{"Integer", "Boolean", "String", "Real"}->
                        union(languagePrimitiveTypes[4]))->
                        includes(self.type.name)
                    else false
                endif
    endif
```

For this rule, the direct translation of the informal specification is also very simple:

```
self.type.oclIsKindOf(Class) or self.type.oclIsKindOf(Interface) or
self.type.oclIsKindOf(DataType)
```

The specification was changed in order to take into account the peculiarities of the Rational Rose tool. In Rational Rose, each type referred to but undefined is represented as an instance of the `DataType` metaclass at the model level. This instance has the name of the type referred in the model. Moreover, in case of late design models, primitive types defined in the implementation language are used. Consequently, all the `DataType` instances having names different from the UML data types or from the target language data types are types referred to but undefined, and have to be reported in the checking process. The specification we used also considers another peculiarity of Rational Rose. Instances of the `Enumeration` metatype are modeled as instances of the `Class` metaclass having the stereotype <`enumeration`>. The enumeration literals are modeled as attributes. The type of these attributes does not matter (in Rational Rose); in many cases it is `undefined`. Therefore, in case of attributes defined in classes stereotyped with <`enumeration`>, the rule `WFR_2_StructuralFeature` must be evaluated to true.

Next, we show that this inconsistency is due to the fact that a referred type was neither defined in the user model, nor imported from the Visual Basic library.

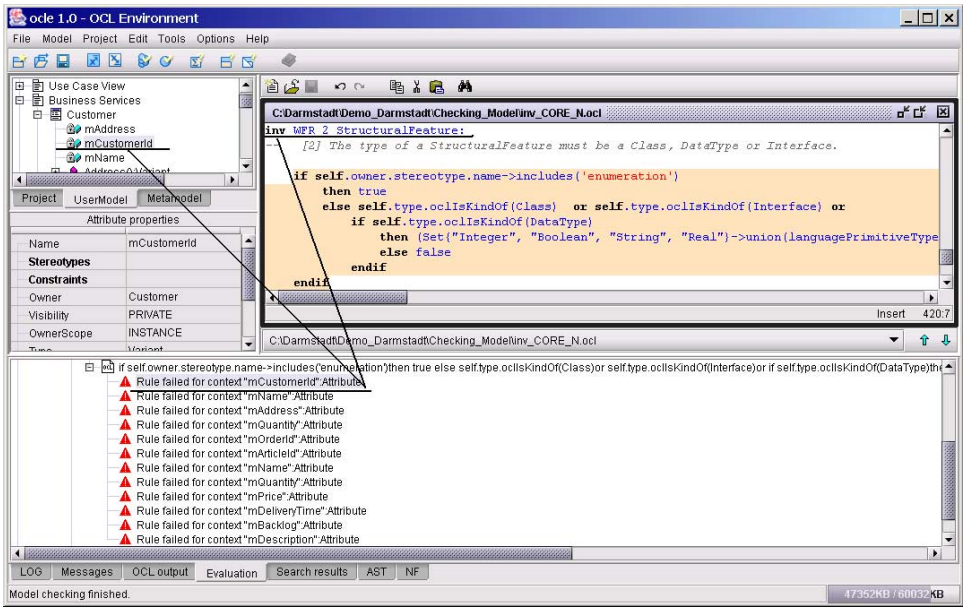

Fig. 3. Attributes having improper type

In the "ordersys" model, the above-mentioned rule is evaluated to `false`

for twelve attributes, all having the type `Variant`. All these attributes are shown in the "Evaluation" pane under the node corresponding to this rule – see Figure 4. Selecting the `mCustomerId` attribute in the "Evaluation" pane, the specification of the rule will be automatically selected in the text editor window, and the corresponding model element will be selected in the "UserModel" browser. By evaluating the attribute type (see single evaluation – Section 3) we notice that this type is an instance of the metaclass `DataType`, named `Variant` – Figure 4.

Analyzing the Visual Basic libraries, we notice that the `Variant` class is included in the package "`stdole Ver 2.0 (OLE Automation)`". Still, this class does not appear in the UML model. In order to solve the problem, a `Variant` class will be created in this package, the type of the above-mentioned attributes will be set to this `Variant` class and the instance of `DataType` named `Variant` will be deleted.

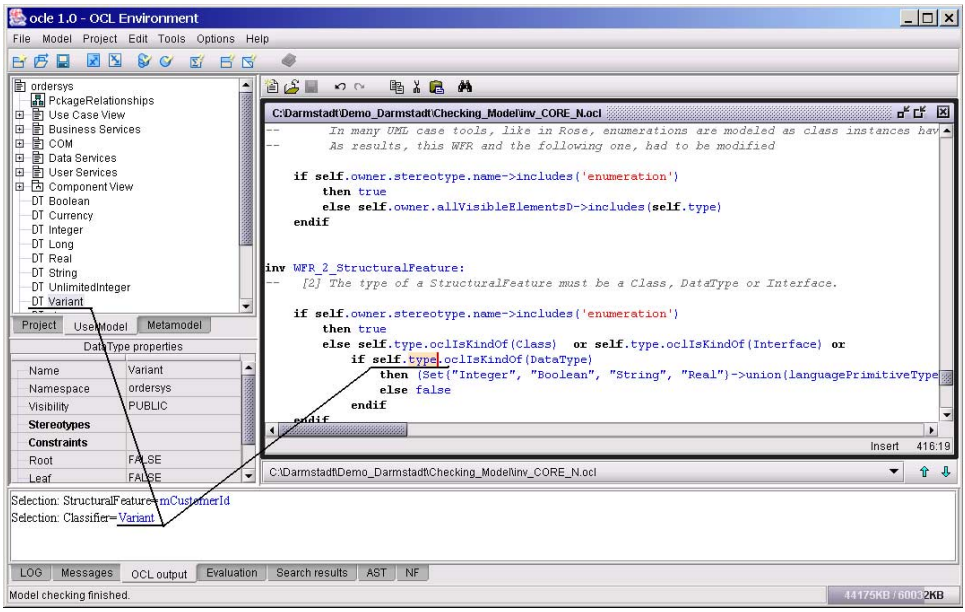

Fig. 4. Fixing the failure cause by evaluating a subexpression

4.2 The UML Crash Course model

This is an example model provided by MagicDraw [13]. The model is smaller than the "ordersys" model. Consequently, the number of evaluations requested and performed with the purpose of checking the well formedness of the model is also smaller (2135). Yet the number of problems found was larger (120). The kinds of inconsistencies found were more diverse. Similarly to the "ordersys"

model, a number of inconsistencies were due to: name conflicts (many model elements with the same name in a namespace), signature conflicts (at least two operations with the same signature in a classifier) and access violations (use of servers without permission). Other types of inconsistencies appeared as a result of: navigable `associationEnd` attached to an `Interface`, parameters without attached type in `BehavioralFeature`, state machines not attached to a context, transitions not associated to an event (trigger). It is worth mentioning that in this second example, a lot of inconsistencies were reported as "evaluation exceptions", the result of those evaluations being undefined. In OCLE, if a rule cannot be evaluated in the context of a model element, the pair (rule, model element) will trigger an evaluation exception.

4.3 A brief conclusion

The models presented in Section 4 were created using two of the most popular tools: Rational Rose and MagicDraw. Using OCLE we highlighted some inconsistencies concerning the well formedness of these models. Most of these inconsistencies are difficult or even impossible to identify using existing CASE tools. Our point of view is that users have to know about them in order to avoid their potential effects.

5 Related Work

MMT [9] is a tool for defining, checking and proving properties about modeling notations. It is based on a simple notion of object that supports a wide variety of modeling configurations. The tool conception contains many interesting ideas. However, the downloadable tool version is an early prototype. Consequently, at this moment the tool is not appropriate for testing real-life models.

The Kent Modeling Framework (KMF) [10] provides a set of tools to support model driven software development. At the core of KMF is ToolGen, a tool to generate modeling tools from the definition of modeling languages expressed as metamodels. KMF supports UML model transfer using XMI. Like MMT, the downloadable tool version is an early prototype and the examples provided with it are small examples. Therefore, the downloadable version is not yet appropriate for checking real-life models.

xLinkit [11] is a lightweight application service that provides rule-based link generation and checks the consistency of distributed web content. xLinkit supports XML technologies. The language used to specify the rules resembles OCL a lot. The rules used to test UML models were translated from OMG's WFR. Since these rules have many drawbacks, the results obtained in checking

the well formedness of UML models are not always correct. The results are provided in a report, and mention only if a rule was evaluated to `false` or `true`. This information is therefore not useful in identifying the causes of any rule failure.

USE [12] is a very interesting UML tool offering OCL support at the model level. Its repository implements a part of the UML 1.3 metamodel. Consequently, using this tool in checking the well formedness of UML models is difficult and restricted to some checks. The tool was used in checking the syntactic and semantic correctness of a part of the UML 1.3 WFR (see [5]). The transfer of models created using other tools is difficult because in USE UML models are stored in a proprietary format.

6 Conclusions

The results obtained in the experiments described in Section 4 confirmed that using OCL in checking UML model consistency represents a valuable approach that is worth taking into account. The models used in these examples were real-life models constructed using known UML CASE tools. OCL offers all the support needed to take into account the peculiarities of different tools (like the modeling of enumerations, undefined types and so on). By using OCLE we succeeded in catching different kinds of inconsistencies that cannot be identified using other approaches (for example name conflicts for `CollaborationRole` instances, undefined contexts). Also, we identified several shortcomings in the UML standard. The lack of a clear rule about the naming of model elements is the simplest example having an important impact on model consistency. The metaclasses whose instances are allowed to be unnamed have to be clearly mentioned. A clarification of what are valid names in UML is also needed. This is of utmost importance for those metaclasses that have correspondents in the target programming language (such as classes, attributes, or even association ends).

All the rules concerning the consistency of UML models are defined at the metamodel level. Therefore these rules are independent of the user model, supporting their reuse for any UML model. This approach is the only one supporting the validation of UML static semantics. This validation activity is very time consuming, because it requires many tests. The results obtained in this domain using OCLE are presented in [1].

Using OCLE we proved that checking the well formedness in case of real UML models is entirely possible. OCL offers the support needed in the MDA approach. In our opinion, all these are strong arguments that OCL is valuable and should become a de-facto industry standard.

References

[1] Dan Chiorean, Adrian Cârcu, Mihai Pasca, Cristian Botiza, etc., *UML Model Checking*, Studia Univ. "Babes-Bolyai" **1** (2002), 71–88.

[2] T. Kielland, J. A. Borrentzen, *UML Consistency Checking*, Research Report, Institutt for datateknikk og informasjonsvitenskap, Oslo, Norway.

[3] Michael Moors, *Consistency Checking*, Rose Architect, Spring Issue, April 2000, http://www.therationaledge.com/rosearchitect/mag/index.html.

[4] C. Nentwich, L. Capra, W. Emmerich A. Finkelstein, *xlinkit: A Consistency Checking and Smart Link Generation Service*, (White Paper), Department of Computer Science, University College London, England, http://www.xLinkit.com/whitepapers.html.

[5] Mark Richters, Martin Gogolla, *Validating UML Models and OCL Constraints*, Proc. 3rd International Conference on the Unified Modeling Language (UML), Springer-Verlag, 2000.

[6] UML 1.5 March 2003 version, 03-03-01.pdf, http://omg.org/technology/uml/index.

[7] Warmer J, Kleppe A., *The Object Constraint Language*, Addison Wesley, 1999.

[8] Warmer J, Kleppe A., *The Object Constraint Language Getting your models ready for MDA*, Addison Wesley, 2003.

[9] MMT Tool, http://www.puml.org.

[10] KMF Tool, http://www.cs.kent.ac.uk/people/staff/sjk/kmf/tools.html.

[11] xLinkit Tool, http://systemwire.com/xlinkit/.

[12] USE Tool, http://db.informatik.uni-bremen.de.projects/USE.

[13] MagicDraw Tool, http://www.magicdraw.com/.

[14] Objecteering Tool, http://www.objecteering.com/.

[15] Together Tool, http://www.borland.com/together/controlcenter/index.html.

[16] OCLE Tool, http://lci.cs.ubbcluj.ro/OCLE.