OCL as a Core UML Transformation Language
WITUML 2002 – Position Paper

Damien Pollet Didier Vojtisek
Jean-Marc Jézéquel
INRIA / IRISA, Campus de Beaulieu
F-35042 Rennes cedex, France
email: {dpollet, dvojtise, jezequel}@irisa.fr

8th April 2002

Abstract

Software developers spend most of their time modifying and maintaining existing products. In a recent article [7] we focused on the definition of UML refactorings (i.e. behavior-preserving transformations) specified by meta-level OCL pre- & post-conditions. In this paper we propose to extend OCL with model modification features, thus allowing to implement model transformations at the same abstraction level as their specification. We then describe the global architecture of a model manipulation tool based on an OCL core.

1 Introduction

The technology of models is receiving more and more attention in the software engineering domain. UML formalism has now evolved enough so the industry is interested in such a standard. However, in the industry, underlying concepts are still not completely introduced in the development lifecycle. One of the key points of model engineering methodologies is the use of technologies for model transformation.

For example, the OMG new proposition about MDA (Model Driven Architecture) consists in the separation of platform dependent and platform independent models. This allows the architect to have a better abstraction in his models for a good reusability. Then, the architect can go from a model to another by applying transformations to it. But soon enough, the transformations can get as complex as the initial model itself, and need to be organized and managed using sound software engineering principles. This is the case for instance in some application domains like real time where PIMs can be very different from PSMs due to heavy constraints on the design.

So, there is a need for a language to write transformations and then the specification of a tool for transforming models. For this purpose, we propose here to extend the OCL language which is integrated with UML and partially standardized. We will also propose a global tool architecture that might be able to implement this approach.

2 UML transformations

2.1 A whole world of transformations

Software design is not limited to the creation of new applications from scratch, and even then, it is not a straightforward process. Designers use multiple transformations of multiple types:

Creational: typically model import from an external source such as XMI or by reverse engineering;

Endomorphic i.e. from UML to UML, such as refinement, refactoring or high-level design;

Exomorphic: from UML to other languages such as XMI or source code.

2.2 OCL description

In [7] we focused on the definition of a special kind of transformations: UML refactorings, i.e. the adaptation to UML of Opdyke's behavior-preserving transformations [4]. We specified each transformation using pre- and post-conditions, expressed as OCL constraints at the metamodel level. This is illustrated in the following example.

Example: attribute privatization This refactoring makes a public attribute from a given class private, and creates the associated getter operation (see fig.1).

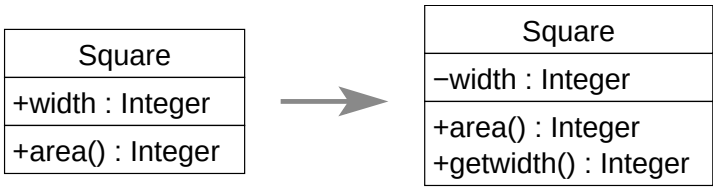

Figure 1: Privatization of an attribute

```
context Class::privatize(a : Attribute)
pre:
  self.feature->includes(a) and
  a.visibility = #public
post:
  a.visibility = #private and
  self.feature->exists(getter : Operation|
    getter.name.body = 'get'.concat(a.name.body) and
    getter.parameter->isEmpty and
    getter.isQuery)
```

2.3 One step further: OCL actions

As of UML 1.4, OCL is restricted to an expression language: while it is quite useful to navigate the model or to express constraints between model elements, it cannot be used to effectively *modify* a model. One of the submissions to the OMG UML/OCL 2 RFPs [1] proposes to extend the expression language with action features. This is particularly interesting as it would allow to define model transformations in OCL along with their pre/post-conditions:

```
context Class::privatize(a : Attribute)
pre: -- ...
action: -- pseudo-OCL syntax for actions
  a.visibility := #private;
  var getter := new Operation in
    getter.name := 'get'.concat(a.name.body);
    getter.isQuery := #true
post: -- ...
```

Why OCL ? Model transformation languages or tools have already been developed, following two main approaches:

- XMI-level transformations based on XSLT,
- dedicated transformation languages.

XSLT is a popular approach, but in our opinion it is too low level: XSLT transformations apply to XML trees, and thus add another level of representation to the manipulated concepts. This is clearly not desirable since expressing model transformations already implies complex reasoning between multiple meta-levels. Some tools address this problem by using a dedicated transformation language, or by which offers higher-level constructs. This language is used to transform models either directly or through an XSLT generation phase.

Dedicated languages address these problems by providing specialized constructs or paradigms; the model is then transformed directly or through compilation to XSLT [5]. Over other dedicated transformation languages, OCL has the advantage of being integrated and used as an expression language in the UML norm. Modeling and transformations can then be done using only one language; more precisely, constraints and specifications will use 'traditional' side-effects free OCL, while methods bodies or model transformations will use the imperative OCL extension[1].

3 UML manipulation architecture

In this section we briefly describe the architecture of an OCL-based model manipulation tool.

OCL interpreter This is the core of the tool: it executes OCL expressions over a given model, and accesses that model either to retrieve information or to modify it;

[1] For designers to easily identify side-effects free ('traditional' OCL) expressions, side effects features should use specific constructs even in the concrete syntax.

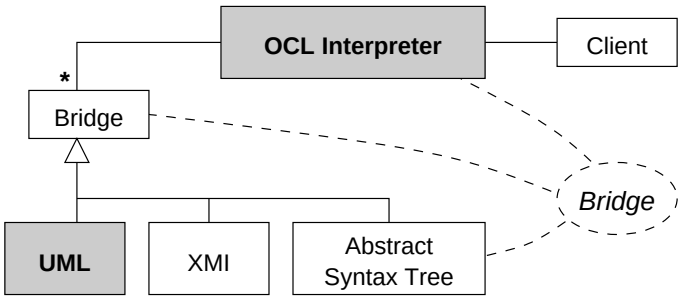

Figure 2: Architecture diagram

UML repository The repository contains the model, represented at the metamodel level (i.e. a class is represented by an object, instance of the M2 concept named Class);

Bridge The OCL interpreter itself should not know anything of UML, but rather manipulate it through a bridge pattern. This bridge maps manipulated MOF-compliant concepts to OCL types and properties.

The bridge is important for multiple reasons:

- it allows easy evolution and interchange of the UML repository, which can be generated from the OMG specification for a given version of UML;
- multiple bridges Implementations can be developed, to enable manipulation of non-UML concepts, such as abstract syntax trees for code generation or XML trees for XMI import/export; these bridge implementations could be seen by the OCL interpreter as libraries for instance;
- multiple repositories can be used simultaneously, enabling transformations between multiple models or notations.

4 Conclusion

Typically, case tools tend to be big and complex software, and to force their users to cope with too much tool-management complexity. We have proposed the extension of OCL to support multiple types of model manipulations and transformations, within a uniform and integrated environment and sketched a tool architecture supporting these ideas. In such a tool, concepts needed to design model transformations are the same as concepts needed to design programs.

As future work, we plan to include structural constructs in OCL, as was done in MMT [3].

References

[1] Initial submission to OMG RFP's ad/00-09-01 & ad/00-09-03. OMG document ad/01-08-35, August 2001.

[2] Xavier Blanc. *Échanges de spécifications hétérogènes et réparties.* PhD thesis, Université Pierre & Marie Curie – Paris VI, November 2001.

[3] A. Clark, A. Evans, S. Kent, and P. Sammut. The mmf approach to engineering object-oriented design languages.

[4] William F. Opdyke. *Refactoring Object-Oriented Frameworks.* PhD thesis, University of Illinois, Urbana-Champaign, 1992. Tech. Report UIUCDCS-R-92-1759.

[5] Mickaël Peltier. Transformation entre un profil UML et un métamodèle MOF. In Michel Dao and Marianne Huchard, editors, *Langages et Modèles à Objets*, volume 8 of *L'Objet.*

[6] Mickaël Peltier, Jean Bézivin, and Gabriel Guillaume. Mtrans: A general framework, based on XSLT, for model transformations. WTUML position paper, 2001.

[7] Gerson Sunyé, Damien Pollet, Yves Le Traon, and Jean-Marc Jézéquel. Refactoring uml models. In *Proceedings of UML 2001*, volume 2185 of *LNCS*, pages 134–148. Springer Verlag, 2001.