RoclET– Refactoring OCL Expressions by Transformations

Cédric Jeanneret and Leander Eyer and Slaviša Marković and Thomas Baar

Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
Laboratoire de Génie Logiciel (LGL)
EPFL-IC-UPLGL, Station 14
CH-1015 Lausanne, Switzerland
Tel: +41 21693 2580 – Fax: +41 21693 5079
e-mail: {cedric.jeanneret,leander.eyer,slavisa.markovic,thomas.baar}@epfl.ch

Abstract: The role of UML models in software development has changed considerably over the last years. While UML was used in its early days mostly as a notation for sketching ideas, developers more and more recognize now the value of a UML model as a formal document that can be processed by tools, e.g. in order to generate code and documentation. Precise software models, however, can usually not be expressed by the pure diagrammatic elements of UML. Instead, they are best captured by constraints written in OCL. Despite the primary importance of constraints within precise modeling, the tool support for OCL is still in a rudimentary stage and there is much room for improvements.

In this paper, we describe the architecture and the functionality of our own OCL tool called RoclET. Currently, our tool supports the parsing of OCL assertions (invariants, pre-/post-conditions), their evaluation in given system snapshots (object diagrams) and refactoring, i.e. the automatic propagation of changes made in the underlying class diagram across annotated OCL constraints. RoclET is highly customizable and can handle different OCL dialects. Its functionality is mainly implemented in form of QVT transformation rules that can be adapted by the user. RoclET is realized in form of an Eclipse plugin.

Keywords: Object Constraint Language (OCL), CASE tool, Refactoring, Precise object-oriented modeling

1. PRECISE MODELING WITH OCL

The Unified Modeling Language (UML) is today the most popular object-oriented modeling language for software systems. UML is in the first place a graphical notation what makes software models easily accessible by humans. UML diagrams can give a good overview on the modeled software system, but there is a lack of expressive power once the details of the software system have to be captured as well. A prevailing practice to resolve this problem is to add comments to UML diagrams and to clarify the intended meaning using natural language. Such informal comments, however, do not alter the formal meaning of the model and are ignored by tools when processing the model, e.g. in order to generate code. Another disadvantage is that it soon becomes a hard and also ambiguous task to read informal comments once the comments are a little bit more complex.

The Object Constraint Language (OCL), see [6] for both an introduction and the language specification, is a textual language with formal syntax and semantics. OCL constraints capture a wide range of details that software developers wish to express in precise software models. The main application scenario for OCL constraints are UML class diagrams. Here, they can express conditions that should be obeyed in each system state (invariants) and contracts for system operations (pre-/post-conditions).

Most of the current OCL tools, e.g. USE [7], Octopus[5], Dresden-OCL Toolkit [3] and OCLE [4] to name the most influential ones, were developed in academia. Whereas almost each tool offers, besides parsing facilities for OCL, a functionality to generate implementation stubs out of UML/OCL models, relatively little effort has been made so far to analyze the OCL constraints themselves, to provide functionalities for automatic constraint simplification, for refactoring, etc. Another weakness of current tools is their lack of flexibility to handle slightly changed versions of OCL, so-called dialects.

RoclET aims at providing facilities for a painless authoring, processing and analysis of OCL constraints. Furthermore, the tool strives for flexibility with respect to the OCL dialect chosen by the user. The user is also free to create his own OCL dialect and to configure RoclET as a support for this new dialect. The main functionalities of RoclET are:

- Parsing and type analysis
- Evaluation of OCL constraints in a given object diagram
- Refactoring of UML/OCL models
- Finding semantical inconsistencies
- Determining changes made in the object diagram that can alter the evaluation of constraints (Impact Analysis)
- Simplification of OCL constraints

2. ARCHITECTURE OF RoclET

We have chosen a 3-layer architecture for RoclET (comp. Fig. 1): presentation layer, application layer and data layer.

The *presentation layer* consists of editors and views for user interaction. Due to a lack of high quality diagram editors we have decided to implement our own editors for class and object diagrams whereas the editor for OCL constraints is (currently) based on the work of [3]. The presentation layer has direct access to the *data layer* where the edited UML/OCL model is stored in a repository as a formal instance of the UML/OCL metamodel.

RoclET's functionalities are implemented in the *application layer*, mainly in form of transformation rules written in QVT. These transformations work on the repository and usually alter it directly. The implementation of some complex functionalities, e.g. the semantical consistency analysis of OCL constraints, uses services offered by 3rd-party tools.

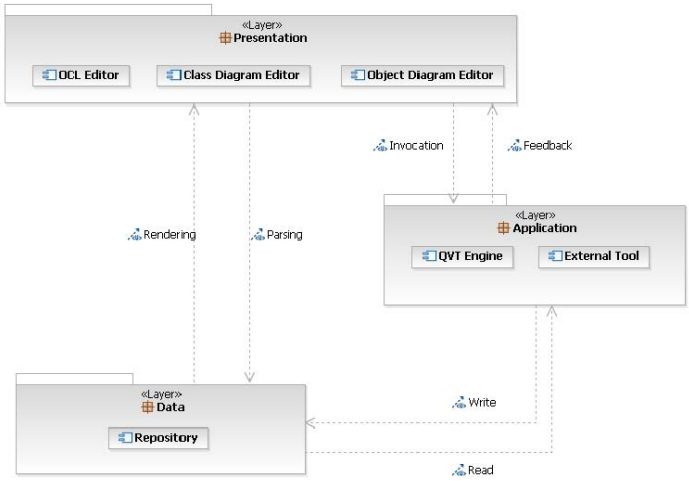

Fig.1: Architecture of RoclET

A unique feature of our tool is its adaptability to specific needs a user might have. Since OCL is basically a very versatile language and applicable in many different domains, there are frequent requests for domain-specific changes of OCL's semantics. It is relatively easy for the user to adapt RoclET to a new OCL dialect. The only thing to be done is to modify some of the QVT rules that implement RoclET's functionalities. In order to do this, however, the user must have installed Together Architect for Eclipse [1], which implements the QVT engine on which RoclET is based on.

3. SYNCHRONIZATION OF ECORE-BASED AND MOF-BASED REPOSITORIES

A major challenge when implementing RoclET was the integration of MOF-based repositories (in our case Metadata Repository (MDR) [2]) with the ECORE-based repository EMF provided by Eclipse. This integration was necessary because the task of parsing/type checking OCL constraints is currently delegated by RoclET to the Dresden-OCL parser [3], which is tailored to MDR as the repository tool. Unfortunately, the Dresden-OCL parser can currently not handle the most recent UML version 2.0 but only UML 1.5 (what has the consequence that (currently) RoclET can only handle UML 1.5 as well).

To our surprise, we were unable to find standard tools for bridging MDR and EMF repositories. We implemented manually a bridge that is optimized for integrating the results of the Dresden-OCL parser into an EMF repository. This bridge is available as a separate component and can be downloaded from www.roclet.org.

REFERENCES

[1] Borland. Together homepage: http://www.borland.com/us/products/together/index.html, 2006.
[2] MDR community. MDR homepage: http://mdr.netbeans.org/, 2006.
[3] Dresden-OCL team. Dresden-OCL homepage: http://dresden-ocl.sourceforge.net/, 2006.
[4] OCLE team. OCLE homepage: http://lci.cs.ubbcluj.ro/ocle/, 2006.
[5] OCTOPUS team. OCTOPUS homepage: http://octopus.sourceforge.net/, 2006.
[6] OMG. UML 2.0 OCL Specification – OMG Final Adopted Specification. OMG Document ptc/03-10-14, Oct 2003.
[7] USE team. USE homepage: http://www.db.informatik.uni-bremen.de/projects/USE/, 2006.