

# OCLLib, OCLUnit, OCLDoc: Pragmatic Extensions for the Object Constraint Language<sup>\*\*\*</sup>

Joanna Chimiak–Opoka

Institute of Computer Science, University of Innsbruck, Austria  
joanna.opoka@uibk.ac.at, <http://qe-informatik.uibk.ac.at/>

**Abstract.** The usage of the Unified Modeling Language in the industrial context becomes increasingly popular. There is an agreement in academia that the Object Constraint Language (OCL) is suitable for defining model constraints and queries. However, it has not yet been broadly adopted by practitioners because they find it difficult to define OCL expressions. Thus, simplification is desirable to increase the use of OCL in practice. We propose OCL libraries (**OCLLib**), which simplify the development of OCL expressions and enable a high reuse factor, are configurable, testable (**OCLUnit**) and documented (**OCLDoc**). In this paper we present the underlying concepts related to OCL library development we used in UML specific and domain specific projects conducted in academic and industrial contexts, respectively.

**Keywords:** systematic development of OCL, OCL libraries, OCL testing, OCL documentation

## 1 Introduction

The Unified Modeling Language (UML) is the well-supported, de-facto standard for object-oriented design and analysis of software systems used to design *large scale models*. The quality management of models can be supported by the use of constraints and queries expressed in the Object Constraint Language (OCL). The maturity of the OCL **syntax and semantics** caused its utilisation within other Object Management Group (OMG) standards and extension of its scope to any language based on Meta Object Facility (MOF). In our recent projects we have successfully used OCL for model assessment and found that OCL is *expressive* [1] and its interpretation is *fast* [2] enough for querying large scale models. In contrast to the syntax and semantics, the **pragmatics** of OCL needs further

---

\* The research herein is partially conducted within the competence network Softnet Austria ([www.soft-net.at](http://www.soft-net.at)) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

\*\* The original publication is available at <http://www.springerlink.com/content/87055v3447758750/>

improvements. Despite the fact that the language became broadly supported by modelling tools [3], practitioners still find OCL specifications difficult to understand [4] and their development difficult, error-prone and time-consuming [5]. The twofold characteristic of OCL (it was designed to be used at the modelling level but has a textual notation closer to the programming level) causes that model designers may find it too formal and programmers too abstract. The positive consequence of OCL being similar to programming languages is the fact that some best practices known from the software development context can be used for development of OCL expressions.

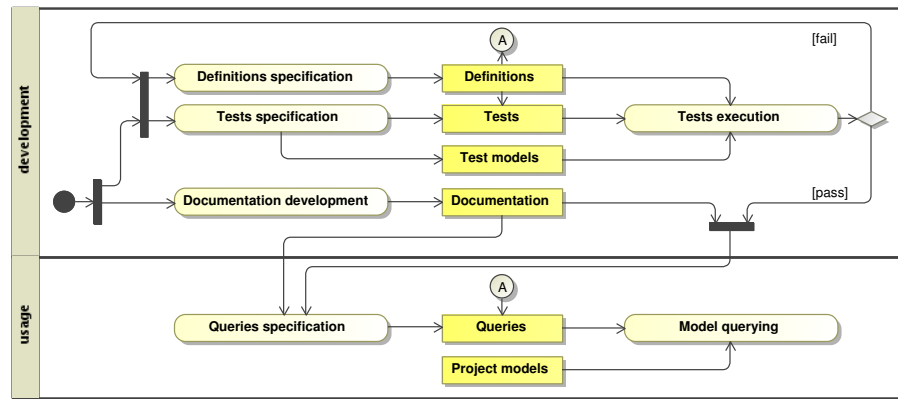
Before we introduce our solution we want to discuss selected **challenges** related to development of OCL specifications. *(C1) Error-free OCL development* is hardly feasible. For the software development no error-detecting approach will ever be able to produce error-free software [6]. For large and complex OCL specifications the same problem is encountered, but it is possible to reduce the number of syntax and semantic errors. As the syntactical correctness is crucial it is already reflected at theoretical and technical levels, but not all issues related to the semantical correctness are solved. *(C2) Easy to understand OCL expressions.* Correct expressions (C1) should be understandable by developers and users [4]. In general, all techniques used in programming languages can be supportive for this challenge, e.g. usage of simple algorithms and data structures, meaningful names, following coding conventions, documentation, tracing and debugging. *(C3) Easy and efficient OCL development* is an idealistic and subjective view. The language itself can not be simplified but its complexity can be leveled by providing learning (C2) and development support. Experience knowledge should be stored and shared. If there are examples of correct OCL expressions (C1) available (which can be customized, or even better, only parametrised) then the development of OCL expressions should be easier and more efficient. For this challenge technical support plays a huge role. *(C4) Easy to evolve OCL expressions.* Similarly, as any piece of code OCL specifications are evolving [4]. There are two critical dangers: the meaning of some parts of the specification can be forgotten and as such hard to evolve or refactor to cover new requirements; and introducing new parts or updating existing ones can have undesired impact of other parts of a specification.

We propose to use the following established techniques: a *systematic development* including usage of libraries, testing and in-code documentation. In the remainder of the paper we present the extended development process (Section 2) and discuss our results and future work in the context of the aforementioned challenges (Section 3).

## 2 The Extended Development Process

In this section we present the idea of testable and documented libraries of OCL expressions and their usage. Depending its purpose, a library can consist of different components: definitions, constraints, queries, tests and their documentation. A collection of libraries and test models forms an OCL library project. Such a

project can be used by another tool, e.g. constraints can be used in a modelling tool and queries can be used in a model analysis tool. An overview of the OCL library project **development and usage process** is illustrated in Fig. 1. The upper swimlane corresponds to the OCL library project **development** stage (Fig. 1). It starts with a specification of OCL *definitions* and *tests*. We denoted these activities as being parallel while different approaches can be used here. One can traditionally start with the definition specification or can follow the test driven approach [7] and define tests and *test models* first. This is up to the developer. As soon as definitions, tests and test models are defined the tests can be evaluated. If the tests return expected results (denoted as [pass] in the diagram), and in the meantime the *documentation* was created, one can start using the definitions. Otherwise (denoted as [fail]) the development process should be continued. As already mentioned, we consider different **usage** scenarios of OCL library project: tested and documented definitions can be used in other definitions, *constraints* or *queries*. In Fig. 1 we present the latter case. At this stage queries are designed and used in model querying on project models. The difference between test and project models is that for the former the expected results are known and test models should not change. If a test model changes then the corresponding tests should be updated to reflect these modifications. Due to the space limitations we give only an overview of the proposed extensions. More details can be found at <http://squam.info/ocleditor/>.



**Fig. 1.** The library development and usage process.

*OCLLib—Collection of OCL Expressions*. The main goal of a **library** is to provide a set of useful and easily reusable OCL expressions (C3). If OCL expressions are split into small chunks, following the modularity and separation of concerns paradigms, then the probability of OCL expression reuse by parametrisation but without adaptation is high. Additionally, as OCL expressions depend on an underlying metamodel (MOF or MOF based) they have to be modularised

into libraries specific to this metamodel, which specifies the scope of application of the library. Furthermore, modularisation based on particular parts of the metamodel can be considered, e.g. libraries specific to UML class diagrams or activity diagrams. To increase and configure reuse, import and visibility concepts are used. A library consists of definitions, constraints, queries and tests which are grouped into blocks, which may be documented. **Definition** and **constraints** are concepts adopted from the OCL/UML standard specifications. Definitions enable better modularisation and higher reuse: defined methods can be used in other libraries, whereas, defined attributes can be additionally used to configure (the parametrisation principle) a library. For example, a collection of metrics can be defined in a library and upper bounds for them in another one, then the metric library can import the configuration library. Both, definitions and constraints can be used by other tools. A **query** is an enhanced concept from the Query/Views/Transformations standard adding parts specific to model querying and as such can be used in model design and analysis tools.

*OCLUnit—Testing of OCL Expressions.* Validation (C1) is required before OCL definitions are used in other libraries or by other developers (C3). In programming practice testing plays a manifold role during the lifetime of a piece of code. Introducing testing to the OCL development practice addresses **all challenges** discussed in Section 1. Testing reduces bugs (C1) and moreover, bugs need only to be found once [8], if they are again introduced due to code changes (C4) they can be automatically detected with prior defined tests. Moreover, a piece of code is usable (C3) for anyone (else) only if it passes all available tests. Additionally, a test case is a simple scenario with a known result, and can be used to understand (C2) code being tested. As pointed out in the problem statement the perceived complexity of OCL is high and testing gives a developer a high degree of confidence that a piece of code is correct. Thus, testing can increase the usage of OCL by practitioners.

*OCLDoc—Documentation of the OCL Expressions.* Documentation of any software artefact is important for many reasons. Among others as a mean to **knowledge transfer and communication**. Moreover, high-quality software documentation reduces the maintenance burden and improves productivity by enhancing reusability. The programming practice [8] showed that the best way to keep a technical documentation up-to-date is to generate it out of source code comments, as it can be written simultaneously with coding and with the same tool by a programmer. Based on documentation OCL developers can easier search for similar expressions and reuse or refactor them, where are OCL users can make a proper choice of needed expressions.

### 3 Discussion and Conclusion

In this paper we analysed challenges for a pragmatic OCL development. We presented three extensions as a possible partial solution to these challenges. We successfully used the described development process in a number of didactic and research projects. The largest project, regarding the size of OCL libraries,

was conducted during the previous semester and had didactic purposes and an evaluation aim. In a period of 2 weeks 10 students developed 50 libraries (4.5kLOC excluding comments) to implement a set of UML metrics [9] in OCL. Even though students had low experience with OCL they found the assigned task easy. The largest project regarding the size of models, was conducted this year within an industrial context. The aim of the project was to document and improve a business process and IT infrastructure. In this project model queries were successfully used to improve the quality of process models (250 entity elements).

The solution we proposed address all challenges at the conceptual (introduced extensions) and at the implementation level (the tool). To address the challenges we currently implement tracing and debugging (C1–C2) and in the future we want to integrate concepts of patterns and to collect and evaluate guidelines for an efficient OCL library development (C3). Another open issues are impact analysis, regression testing and refactoring support (C4).

**Acknowledgement** I want to express my gratitude to Barbara Weber and Berthold Agreiter for numerous discussions and their constructive feedback, and to Nicolas Rouquette and Dan Chiorean for their feedback and improvements ideas for our tool. Moreover, my gratefulness goes to all members of the OCL editor development team for their great cooperation in tool development and discussions on underlying concepts.

## References

1. Chimiak-Opoka, J., Lenz, C.: Use of OCL in a model assessment framework: An experience report. *Electronic Communications of the EASST* **5** (2006)
2. Chimiak-Opoka, J., Felderer, M., Lenz, C., Lange, C.: Querying UML Models using OCL and Prolog: A Performance Study. In: *Model Driven Engineering, Verification, and Validation*, Lillehammer, Norway (4 2008) presented at MoDeVVA.
3. Baar, T., et al.: Tool support for OCL and related formalisms - needs and trends. In Bruel, J.M., ed.: *MoDELS Satellite Events*. Volume 3844 of *Lecture Notes in Computer Science*, Springer (2005) 1–9
4. Correa, A.L., et al.: An empirical study of the impact of ocl smells and refactorings on the understandability of ocl specifications. In Engels, G., et al., eds.: *MoDELS*. Volume 4735 of *Lecture Notes in Computer Science*, Springer (2007) 76–90
5. Ackermann, J.: Fallstudie zur spezifikation von fachkomponenten. In Turowski, K., ed.: *2. Workshop Modellierung und Spezifikation von Fachkomponenten*, Bamberg, Deutschland (2001) 1–66 (In German).
6. Glass, R.L.: Two mistakes and error-free software: A confession. *IEEE Softw.* **25**(4) (2008) 96
7. Beck, K.: *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
8. Hunt, A., Thomas, D.: *The pragmatic programmer: from journeyman to master*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1999)
9. Genero, M., Piattini, M., Calero, C.: A survey of metrics for uml class diagrams. *Journal of Object Technology* **4**(9) (2005) 59–92