

# Evolution of the OCL OMG Specification

Mariano Belaunde<sup>1</sup>

<sup>1</sup> Orange Labs, 8 Avenue Pierre Marzin  
22300 Lannion, France  
mariano.belaunde@orange-ftgroup.com

**Abstract.** The OCL language has reached a good acceptance in the model-driven community and a variety of commercial or academic tools support the language since many years. However, maintenance of OMG OCL specification is painful for various reasons. This paper describes the process for maintaining the specification and explains some of the reasons the evolution of OCL requires a big amount of work. It also describes major issues that the revision and finalization task forces need to solve and the major changes introduced in the specification. Finally, some perspectives on future evolution of OCL standard are provided.

**Keywords:** OMG, OCL, UML, Modeling

## 1 Introduction

The OCL language (Object Constraint Language) has reached a good acceptance in the model-driven community. A variety of commercial or academic tools support the language since many years. OCL is often used by modelers to complement structural diagrams with useful information that enhances the level of precision of models, encompassing the limitations of entity/relationship paradigm. OCL is also used to define declaratively the effects of an operation on an object. Another noticeable application of OCL is its usage as a query language for models, for which OCL provide interesting constructs. Its role as a query language has motivated its reuse in other OMG standards, such as QVT for model to model transformation and Mof2Text for model to text generation.

The OCL specification depends on various other standards such as MOF and UML which have evolved significantly since 2005 with the adoption of major evolution of UML (going from 1.x series to 2.x series). In fact the OCL specification quickly becomes de-synchronized in respect to UML2. Maintenance of OMG OCL specification is painful for various reasons that will be explained in this paper. However, despite the slowness of the revision process, most of fundamental changes have been achieved to make OCL usable in the context of modern modeling tools and frameworks.

Section 2 gives an overview of OCL standardization process and the structure of the specification, Section 3 focus on the reasons that make OCL specification hard to maintain, Section 4 explain some of the most relevant changes introduced by OCL. Finally the last section gives some perspective and conclusions regarding future evolution of OCL.

## 2 Overview of OCL standardization

The OCL language definition was initially included as part of UML 1.3 specification (1999). Then an important revision was adopted as a draft in 2003. Unfortunately this revision, although labeled as 2.0, arrived too early in respect to UML 2.0 and MOF 2.0. As a consequence, most of diagrams and defined constraints were done in terms of UML1.3 vocabulary. Finalization of OCL 2.0 was completed in 2006, but only certain parts of the specification were effectively aligned with new terminology and structure of UML2. The revision task force continued the work and produced a new revision numbered 2.2 in 2009 (the 2.1 revision was skipped).

The OCL Revision Task Force plans to deliver next revision 2.3 of OCL specification in December (presentation to the Architecture Board). We anticipate a shift of three months. Official release come generally between three or six months later after AB review, passing formal validation and final editing steps.

The OMG process for revising a specification is relatively *heavy* but guaranties openness (any person can post issues for resolution), fairness (clear rules to allow original submitters and supporters to keep control on the evolution) and transparency (all changes introduced by the task force can be tracked and are subject to vote).

Typical work of the revision task force is summarized below:

1. Look at the issues database. Actually 110 issues deferred by OCL 2.0 finalization and 340 issues submitted to the revision task. 84 issues were solved in OCL 2.2. Percentage of typos and clarification issues is 30%. We found some level of duplicated issues (10%). Any individual - not necessarily an OMG member - can post issues. Members of the revision task force indeed should be OMG members.
2. Select a set of issues and propose resolutions for each issue (Resolved, Closed, Deferred, Duplicated and Transferred). Resolution contains the detailed instructions for an independent editor to be able to apply the changes to the specification. The main difficulty is to capture the impact of a modification in all the sections of the specification.
3. Edit the specification framework document with change bars to apply the proposed changes. Explicit marks need to be inserted in the document to allow a reviewer to know the issue originating the change. For timing

reasons, this editing phase is often deferred to after the adoption of all resolutions (which is not an ideal situation). Usage of *framemaker* format also poses license problems: it is not free, it is very expensive and the evaluation version is only valid during one month.

4. Organize a ballot vote. Good practice is two weeks to examine the proposed resolutions before starting the vote.
5. Iterate over this process until having solved a reasonable quantity of issues.
6. Compile the revision report including all resolutions and the general statements. Present the report to the architecture board (AB). Implement any changes requested by the AB to validate the report.
7. An OMG editor makes the final formatting of the document and publishes it.

### **3 Main obstacles to OCL specification maintenance**

There are several reasons that make the maintenance of OCL specification painful. In short the two main reasons are explained below:

- Companies are less interested in maintaining already adopted specifications than creating new ones, especially when the revision demands a big investment. Between 2003 and 2005, OCL specification was almost left "orphan" with risk in 2005 for the OMG to be obliged to withdraw a non-finalized OCL 2.0 specification. OCL maintenance was in fact taken in hand by QVT submitters. But after QVT 1.0 finalization in 2007, it become very difficult for companies to continue to allocate man power for OCL maintenance.

- The OCL 2.0 draft was very ambitious (probably too much!) in terms of formality. For instance, it was the first specification attempting to define formally the mapping between the textual syntax and the metamodel. This was certainly a very good idea; however, the problem for the team in charge of the finalization was that there was no tool to check consistency and real effect of the mapping expressions. Formality requires tools. Without tools for checking formal definitions it is very difficult to trust persons written formal expressions. This is due to high probability to make mistakes.

Apart from that, the specification also contained other formalizations that were much more questionable in terms of an effective added value. One of them was the attempt to formalize the semantics by creating a specific metamodel representing OCL semantics, in parallel to the one used to define the abstract syntax. This introduced excessive amount of models and text to be verified with unclear benefits for implementers and users of the language.

Despite these obstacles and thanks to some enthusiastic OCL experts (see acknowledgment section), OCL evolved, indeed not as fast as wanted, but evolved.

Notice that other specifications like MOF 2.0 Core, despite its central role within OMG modeling standards, and despite its relative reasonable size (only 88 pages, versus 232 for OCL) have difficulties to be updated. The new specification MOF 2.4 Core is arriving in 2010 after 5 years of completion of MOF 2.0 finalization.

In Section 4 we mention an ongoing initiative to *agilize* significantly the maintenance of the specification.

## 4 Major changes in OCL specification

In this section we summarize changes done to OCL since last years.

From the *draft* OCL 2.0 specification in 2003 [1] and the *finalized* OCL 2.0 in 2006 [2] the following important changes were made:

- *Essential OCL*: A variant of OCL to make it usable not only in the context of UML but also for other metamodels. The Essential OCL metamodel - which is an EMOF compliant metamodel - removes concepts that are specific to UML (like State and Message notions) and aligns metaclass hierarchy with EMOF metaclass hierarchy (using Type instead of Classifier as the base class for types)
- *Distinction between Null and Invalid*: In former OCL, only the notion of *undefined* was defined. As a consequence, an OCL specifier could not differentiate between absence of value in an association and a faulty access (like division by zero). QVT influenced in this change despite some resistance.
- *Adjusting OCL metamodel* so that it refers to UML2 concepts instead of UML1.4 concepts (like Attribute replaced by Property).

Most of these *major* changes done by the finalization task had a strong impact in all parts of the specification. However, for timing reasons, only essential chapters considered crucial for implementers were updated (abstract syntax, OCL library and the grammar).

From the *finalized* OCL 2.0 specification and the *new version* OCL 2.2 in 2009 [2] the following important changes were made:

- Removal of *special* enumeration types to represent definitions at different level of abstraction). Instead explicit access to a model representing the meta-level is used.
- OCL Collections are full objects

- Enhancing the OCL pre-defined library with additional utility functions.
- Fixing multiple pending inconsistencies left by the finalized OCL 2.0.

## 4 Perspectives

The maintenance work for OCL is far from being complete. Very formal parts like the mapping between abstract and concrete syntax (Chapter 8) and the OCL semantics (Chapter 10) are still not fully verified and aligned with OCL abstract syntax. Also, regarding the Essential OCL flavor, only the metamodel was specified, not the impact of the specialization on well formed-ness rules and OCL concrete syntax.

But besides that there are other fundamental problems that are still waiting for a solution:

- Lack of user-defined parameterized types (opposed to pre-defined parameterized types like the existing OCL collection types). In fact the existing OCL library cannot be properly represented as an instance of the OCL metamodel without using some hacking turnarounds (like the `TemplateParameterType` class).
- Lack of the notion of *contextual operations* as extensions of metaclasses (like in QVT Operational) to allow adding operations to pre-existing metaclasses. The implicit assumption in OCL is that any *helper* operation should be part of the original definition of a class. This creates inconsistencies: it is impossible to foresee all helper operations that can be needed.

Other pending problems and issues have been identified and are worth to mention. The list indeed is not exhaustive:

- Overloading and dynamic dispatch of operations in OCL
- Support of stereotypes
- Incompatibilities between MOF, UML and OCL primitive types
- Possible conflict between '.' and '>' operators for OCL collections since 2.2.
- OCL reflection (a meaning for 'self.oclType().oclType().ownedOperation' ?)

We mentioned in Section 3 some difficulty to maintain parts of the OCL specification that are formalized with ad-hoc techniques, without an effective tool support. Since new standards like QVT and Executable UML have arrived and have tool support, some of the considerations for the future of OCL revision could be:

---

<sup>1</sup> According to analysis made by Ed Willing (Thales)

- Re-specifying the mapping between concrete notation and abstract notation with QVT
- Re-specifying the OCL semantics as a transformation to executable UML.

Now concerning the major problem of how to make OCL specification maintenance agile, we have started to investigate possible setup of a semi-automated procedure for producing the documents that are required by the OMG process (issue resolution reports, *framemaker* files with and change bars) on the basis of code generation. The basic idea will be to model relevant parts of the specification (initially populated by retro-analysis of existing framemaker files) and then express issues resolutions as changes to the model of the specification. If we succeed in this task, the main advantage will be to improve the overall consistency of the document, and hence limit the amount of "stupid" issues (typos, forgotten updates and so on), so that man power can concentrate in the resolution of fundamental issues.

## Acknowledgements

OCL evolution was made possible thanks to the involvement of many persons that it would be too long to cite here. In recent years, some people were particularly active in proposing resolutions. I will hence particularly mention Christian Damus, Ed Willink and Adolfo Sanchez-Barbudo Herrera.

## References

1. Object Management Group (OMG); *Object Constraint Language Specification*, Chapter 7 of *OMG Unified Modeling Language Specification*, Version 1.3, March 2000 (first edition).
2. Object Management Group (OMG); *Object Constraint Language OMG Available Specification Version 2.0*, May 2006