Chapter 1

THE DRESDEN OCL TOOLKIT AND ITS ROLE IN INFORMATION SYSTEMS DEVELOPMENT

Birgit Demuth
Dresden University of Technology, Department of Computer Science,
D-01062 Dresden, Germany
Birgit.Demuth@inf.tu-dresden.de

Abstract The Object Constraint Language (OCL) is a part of the Unified Modeling Language (UML), the OMG standard in modeling of object-oriented applications. It is a formal language for defining constraints on UML models, especially in class and statechart diagrams. OCL does not require a mathematical background and can be understood by most software developers. The language is very powerful because it can be used together with class and other UML diagrams at different model layers. By the specification of OCL constraints at the metamodel layer, it is for example possible to ensure the consistency of conceptual modeling artifacts. In the practice of database and software engineering, however, OCL is not yet known because most UML tools do not support OCL. Therefore, at the Dresden University of Technology, we developed a toolkit handling OCL constraints. We also gained with our toolkit initial experience in using OCL in real projects. In this paper we outline the structure of the Dresden OCL Toolkit and how to apply it in information systems development.

Keywords: OCL, UML, database engineering, software engineering, business rules, modeling guidelines, information systems

1. Introduction

The Object Constraint Language (OCL) is a part of the Unified Modeling Language (UML), the OMG standard in modeling of object-oriented applications 18, 2. It is a formal language for defining constraints in different UML models of any domains. The language is very powerful because it can be used at different modeling layers. For example, at the model layer a business modeler can use OCL to specify business constraints, and at the metamodeling OCL

has shown its helpfulness to define the UML semantics much clearer and unambiguous than only by definition with graphical annotations of UML. Over the last years, the "graphical core" of UML has been accepted by the bigger part of software developers as a visualization and documentation tool in analysis and design. An evidence for it is the availibility of a good deal more of hundred UML tools (see UML tools pages such as 21, 22). However, the formal language OCL has predominantly remained an academic subject 14. We think that one reason for this situation is the absence of adequate tools. As opposed to UML tools we know only one dozen of academic and commercial UML/OCL tools. Furthermore, the formal language OCL incites fear in the users to learn and apply such a language. However, we are sure that even for those with no experience in formal methods OCL is learnable and adds precision and detail to software models. For a better understanding a simple example should be given. Suppose there is a (sub)model of a hotel reservation system with the classes `Destination`, `Region` and `Hotel` (see fig. 1.1). A business rule specified in OCL is that the region in which a destination X is located (`regionOfDestination`) must be the same like the region of all hotels (`myRegion`) that are located in destination X. The respective OCL expression is given below:

```
context Destination inv iRegion:
hotelsInDestination->
forAll(myRegion = self.regionOfDestination)
```

The most comprehensive source for learning OCL and and its powerful features for UML-based modeling in practice are the books from Jos Warmer and Anneke Kleppe 12, 13.

In Section 2 we outline the modular structure and openness of the Dresden OCL Toolkit including its possibilities for the integration into other tools. Related tools are listed in section 3 together with a short comparison to the Dresden OCL Toolkit. Finally, we give an overview for OCL use cases in the information system (IS) development and show by a few examples how the Dresden OCL Toolkit can be applied for different purposes.

2. The open source project Dresden OCL Toolkit

2.1 Objectives

In this section we will briefly introduce a software platform for OCL tool support which is designed for openness and modularity, and which is provided as open source 24. The goal of this platform is, for one thing, to enable practical experiments with various variants of OCL tool support, and then, to provide an

OCL library (under the LGPL license 23) for UML tool builders which want to support UML by the specification and evaluation of OCL constraints.

The development of the Dresden OCL Toolkit has been driven by the following objectives 8:

- The architecture shall enable interworking with various UML tools and repositories, regarding the access to model information for typechecking. A simple and flexible interface is required which supports the construction of stand-alone experimental tools (working e.g. on a XMI[1] file representation of the model) as well as a tight integration into UML tools, for more user-friendly versions of tools.
- Syntax analysis and type checking of OCL constraints is the functionality which is common to all tool variants. So a simple interface to this functionality is needed in order to enable integration into various OCL tools.
- The tool platform has to provide a simple and easily reusable interface for accessing the actual constraint information (the abstract syntax of the constraints) from different kinds of tools.
- Different tools need different levels of abstraction in accessing the representation of OCL constraints. For example, a tool generating programming language code may need to expand automatically all `select` operations into the generic `iterate` mechanism. In contrast, a tool generating SQL integrity conditions may need to keep the `select` operations since they can be mapped easily and directly to SQL 4.

The Dresden OCL Toolkit is developed in Java because of the high popularity of Java as an implementation language in the Open Source Community and therewith the availability of useful tools like parser generators and the possibility to integrate the OCL toolkit with free UML tools such as ArgoUML. Its compliance to the above listed requirements has been proven by its multiple reuse both in academic projects (see e.g. the open source tool ArgoUML 26 and the KeY project 28) and commercial tools (e.g. Poseidon 27).

2.2 Structure

The Dresden OCL Toolkit provides by a modular architecture the following tools:

OCLCore: The base tool of the OCL toolkit consists of different modules:

- The **OCLParser** transforms the input OCL expression into an abstract syntax tree 8. The abstract syntax tree classes can be seen as a representation of a static UML metamodel.

- The **OCLEditor** is a comfortable editor which includes besides editing of constraints features like a toolbar and adequate error messages. The according user interface is designed to integrate the OCL editor not into a specific UML tool, but into various environments. The screenshot in fig. 1.1 gives an impression of the OCL editor integrated into Together.
- The **OCLTypeChecker** checks the semantic correctness with reference to the OCL type system and offers type information about the associated UML model towards other modules. The model information has to be extracted from the toolkit's environment. For this purpose a small external interface (called `ModelFacade`) is provided 8.
- The **OCLNormaliser** transforms the abstract syntax tree into a *normal form* of OCL terms, such that all terms can be mapped into a simpler subset of the OCL language. That way it can be avoided that every tool using OCLCore has to implement the execution of any OCL expression completely.

OCL2Java: This tool transforms the normalised syntax tree into Java Code. It uses a class library which offers Java representations for the predefined OCL types.

OCLInjector4Java: In order to make the generated Java code useful, a separate tool (OCLInjector4Java) is required which takes this code and inserts it into the application program. This code instrumentation is done by the generation of wrapper methods for all methods that have to be checked in compliance with specified OCL constraints during execution. The used technique including code cleaning is described in 25. The OCLInjector4Java has been integrated for example into ArgoUML.

OCL2SQL: The SQL code generator 5 generates a SQL check constraint, assertion or trigger for an OCL invariant based (like in the case of Java code generation) on the parsed, typechecked and normalised OCL expression. OCL2SQL can be used and adapted for different relational database systems and different object-to-table mappings. To make this generator work, we need some additional information about the underlying object-to-table mapping. Since there is a great number of different object-to-table mappings 1, an interface is provided for the integration of various strategies.

OCLInterpreter: A first tool developed outside of the Dresden University of Technology is an OCL interpreter that allows the stand-alone checking

Figure 1.1. OCLCore integrated into Together

of OCL constraints against objects. The OCLInterpreter is also designed based on the normalised abstract syntax tree.

OCL20: Currently we reengineer the Dresden OCL Toolkit according to the new requirements of the revised and approved specification of OCL ("OCL 2.0" 17). The OCL20 module is a prototype of a metamodel-based OCL compiler consisting of a MOF 16 repository implementation and a code generator 9. The OCL 2.0 parser is still under development. The research issue is to which extent a parser can be automatically generated from the provided specification.

2.3 Integration of OCL into a UML tool

An important requirement of tools supporting OCL is their cooperation with UML tools. The specification of OCL constraints without any model makes no sense. In our approach the `ModelFacade` has been implemented in different ways. An OCL tool can be **tightly** integrated into a UML tool as an add-in. Then the model interface must be implemented by an integration component accessing the UML tool's repository. Examples for this technique is the integration of our toolkit into Together (see fig. 1.1) Together 36, ArgoUML 26, Poseidon 27, and Rational Rose 37. A kind of **loose** integration is the use of

XMI files 20 for static UML model information. The Dresden OCL toolkit already provides the necessary component to use this technology.

3. Related work

As already noted above, OCL is still an academic subject. There is little knowledge about using OCL in real projects. However with the matter of fact that OCL has become a solid and standardized part of UML 18, 19 more UML tools support the specification and evaluation of constraints with OCL. Most of existing OCL tools implement one of the 1.x versions and dialects, respectively. First tools that promise full support of OCL 2.0 are OCLE 29 and Octopus 30.

In the following we give a short overview about the most common tools that do not reuse the Dresden OCL Toolkit:

OCLE 29: The OCL Environment (OCLE) is a stand-alone tool that can load, edit and save UML models as XMI files. It was developed at the BABES-BOLYAI University (Romania) and can be downloaded for free.

Octopus 30: The Octopus tool from Klasse Objecten exclusively runs as an Eclipse plugin 38 and is distributed under a public license. It is able to check the syntax and semantic correctness of OCL expressions. The current problem using Octopus is the missing model export/import by XMI. Octopus provides proprietary solutions for using it together with Rational Rose and Poseidon.

Bold for Delphi 34: Bold for Delphi is a commercial product for model based development including a runtime OCL interpreter. It uses OCL for different purposes, e.g. to perform queries in the object layer and to define derived attributes.

MagicDraw UML 33: MagicDraw UML is a commercial UML tool that suppports OCL. OCL can be used in an orthogonal way for the specification of invariants, guards, pre and post conditions and as a navigation language. However, the user can only specify and parse constraints. The semantic (type) correctness is not checked.

UMLAUT 35: UMLAUT (Unified Modeling Language All pUrposes Transformer) is a generic model transformation tool that allows one to create transformations for any model for which there is a metamodel. It supports the design-by-contract technique by allowing the designer to specify constraints on models with OCL. The constraints can be parsed and transformed into Eiffel code.

USE 32: USE (UML-based Specification Environment) is a research prototype that supports OCL constraint specification as well as evaluation for

simulated objects. The tool is appropriate for checking the syntactic and semantic correctness of specifications in the research field (e.g. of well-formedness rules of the UML metamodel). The difficulty for the practical use is that there is no XMI model import/export. The UML models have to be specified in a proprietary textual format and objects have to be manually created.

KMF 31: The Kent Modeling Framework (KMF) includes an OCL Library that is still under development.

To sum up, we can state that most of the OCL tools are still at the prototype stage because they either are developed for specialised research studies or they only check the syntax and semantics (sometimes only the syntax) of constraints. In our view the Dresden OCL Toolkit is the only OCL implementation that has a modular architecture with cleanly defined interfaces for the integration into Java-based tools so that OCL can be used for different purposes. And it is by the distribution under an open source license available for multiple reuse.

4. OCL use cases in the IS development

4.1 Overview

The Object Management Group (OMG) understands information management as the design, implementation and mangement of large bodies of more or less structured information 16. So from a technical point of view, information system development matters the application of **software** and **database development methods**. And basically, this can be done in a **forward** or **reverse engineering** manner depending on whether the information system has to be developed from scratch or an existing information system has to be reengineered. A further dimension is the choice of the model layer in terms of the MOF Four Layer Metadata Architecture 16. In our discussion the **metamodel** (M2), the **model** (M1) and **information** (M0) layer should be considered. Inside this frame many OCL use cases are imaginable.

The intention of the first OMG version of OCL 18, 12 was to define a language for a clear and unambiguous specification of things that often cannot be expressed in a diagram (see our OCL example above). In the context of information systems such things particularly are **business rules** that represent business knowledge and govern how the business processes should be executed 6. Their practical importance is considered in 7. In 1994 when OCL had not yet been published, the authors point out that a specification language is needed as a supplement to graphical representation because diagrams for business rules for IS of realistic size become too complex and cumbersome. Now we see into OCL as an OMG standard that it fills up this gap. Table 1.1 summarizes what role OCL can play for the treatment of business rules in the MOF layer archi-

Table 1.1. Using OCL in the IS development.

MOF layer	**OCL constraints in IS**
M2 (metamodel)	Specification of modeling guidelines
M1 (model)	Specification of business rules Evaluation of modeling guidelines on business models Checking consistency of business models
M0 (information)	Evaluation of business rules on business objects

tecture. In the context of a specific IS domain it is often requested to establish domain, project or company specific modeling guidelines that can be specified with OCL at the metamodel layer and enforced at the model layer by evaluation of the specified OCL constraints 11. In 3, it is shown by two practical sample business models that it is even (still) necessary to check their UML model consistency by an OCL tool[2]. Given a (with modeling rules) enriched metamodel or taking the "pure" UML meta model, business rules can be specified by OCL invariants or pre and post conditions at the model layer. Then the business rules have to be evaluated at the information layer represented by the real business objects.

The understanding of OCL 2.0 is that far more additional information should be included in a model than constraints alone 13. In UML2 (whose part OCL 2.0 is) the user can write any expressions on the elements in the diagram. Every OCL expression represents a value or object. Therefore, the use cases of OCL have grown considerably as opposed to the first versions of the language 13:

- definition of derived attributes and associations
- specification of an initial value of an attribute or association role
- specification of query operations
- definition of derived classes
- specification of dynamic and optional multiplicity

That means, OCL is a **constraint** and **query language** at the same time. Database people could argue that this approach is very similar to SQL 10. However OCL is designed for object-oriented modeling, and UML/OCL can be orthogonally and in a standardized way used for all software models, all the same if there are parts that are implemented by a relational data base. At this point, it should be emphasized that the question, how to implement a UML/OCL business model, is an implementation issue. Therefore we need

Table 1.2. Case studies with the Dresden OCL Toolkit.

	Java applications	**Relational Databases**
Forward Engineering	nxCom (JEFF)	till now only academic studies 5
Reverse Engineering	experiments with source/byte code and runtime analysis 25	Entre2Mers

code generators for different languages, basically both for programming and database languages.

4.2 Examples

The case studies using the Dresden OCL Toolkit that are listed below show how the Dresden OCL Toolkit helps to provide the required flexibility. According the above outlined frame of OCL use cases, the examples can be classified how it is presented in table 1.2.

nxCom is a commercial product providing a business directory service. During the development of the nxCom business logic module the OCL2Java and the OCLInjector4Java tool were tested 25. The nxCom module was instrumented with Java code that checks invariants (mainly business rules) against test data. So many constraint violations could be detected.

Entre2Mers was a project of the Electricite de France (EDF). In a EDF data warehouse, many historical databases should be reconstituted. This issue raises database quality problems of different categories like different name and addresses of the same person in different databases. In order to gain more confidence in information contained in relational databases, a database quality analysis tool was developed. After experiments with an own language, OCL was used to express the constraints. In order to implement it, the OCL2SQL module was integrated into the EDF database quality tool.

JEFF is a framework for building business applications developed by sd&m. In 15 OCL was used to specify constraints (both invariants and pre and post conditions) on JEFF classes and components. Although the complexity of such specifications can grow very rapidly, it is possible that a (right) use of OCL can increase the quality of code and the performance of the software development process. Note that the JEFF specification is based on OCL 2.0 and thus could still not be checked with Dresden OCL Toolkit modules.

5. Conclusions and Future work

We presented a modular architecture for the integration of various OCL tools into different UML environments. In the light of the open source distribution of the Dresden OCL Toolkit, this flexibility can be increased by its adaptation and advancement for special purposes. The Dresden OCL Toolkit has already demonstrated its usefulness in 46 (known) projects. Furthermore, we could report about first own experiments with our OCL tools. We found use cases in software and database development as well as in forward and reverse engineering scenarios at different model layers.

Our plans for future work include the further development of the Dresden OCL Toolkit, and then, the practical use of our toolkit in case studies to gain further experience with UML/OCL-based modeling. In particular, the next step is the complete implementation of a metamodel-based compiler with full compliance to OCL 2.0. Such a tool support will allow us to experiment with OCL in the Model Driven Architecture (MDA) framework 13. This particularly includes model transformations and code generation techniques. In the context of information systems, we see a high potential of OCL in data integration scenarios. On the one hand, integrity constraints can be used to extract information of incomplete sources, and on the other hand, constraints can detect inconsistencies of the whole system (see the Entr2Mers project). Such a scenario is also driven by the need to integrate data sources on the web. So OCL could provide potential for the development of the semantic web.

Acknowledgments

I would like to thank all people who have contributed over several years to the Dresden OCL Toolkit project. Heinrich Hussmann initiated the project in 1999. His idea was to bridge the gap between formal methods in software engineering and the practice of software development. In the following years many students accounted both with research ideas and implementations to the Dresden OCL Toolkit and made their modules available to the open source community. Particularly, Frank Finger, Ralf Wiebicke, Steffen Zschaler, Sten Loecher, Stefan Ocke and Christian Nill substantially contributed to the OCL tools existing today.

Notes

1. The XML Metadata Interchange Format (XMI) is an OMG standard for a Stream-based Model Interchange format. The main purpose of XMI is to enable easy interchange of data and metadata between UML modeling tools and between tools and metadata repositories in distributed heterogeneous environments.
2. For the future we hope that UML tools do it in a comprehensive manner by themselves.

References

[1] Blaha, M., Premerlani, W.: Object-Oriented Modeling and Design for Database Applications. Prentice Hall, 1998

[2] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, 1999

[3] Chiorean, D. et al: Ensuring UML models consistency using the OCL Environment. in: [14]

[4] Demuth, B., Hussmann, H.: Using OCL Constraints for Relational Database Design. in: UML'99 The Unified Modeling Language, Second Int. Conference Fort Collins, CO, USA, October 1999, Springer, 1999

[5] Demuth, B., Hussmann, H., Loecher, St.: OCL as a Specification Language for Business Rules in Database Applications. in: Fourth International Conference on the Unified Modeling Language (UML 2001), Toronto, Canada, October 1-5, 2001

[6] Eriksson, H.-E., Penker, M. Business Modeling with UML. Business Patterns at Work, John Wiley & Sons, Inc., New York, 2000

[7] Herbst, H. et al, The specification of business rules: a comparison of selected methodologies. in: Methods and Associated Tools for the Information System Life Cycle. Elsevier, Amsterdam, 1994

[8] Hussmann, H., Demuth, B., Finger, F.: Modular Architecture for a Toolset Supporting OCL. in: UML'2000 - The Unified Modeling Language. Advancing the Standard, Third Int. Conference York, UK, October 2000, Springer, 2000

[9] Loecher, St., Ocke, St.: A Metamodel-Based OCL-Compiler for UML and MOF. in: [14]

[10] Melton, J., Simon, A.: Understanding the New SQL: A Complete Guide. Morgan Kaufmann, 1993

[11] Ritter, N., Steiert, H.-P.: Enforcing Modeling Guidelines in an ORDBMS-based UML-Repository. in: Challenges of Information Technology Management in the 21st Century. Proc. International Resource Management Association Conference IRMA'2000, Anchorage, Alaska, Mai 2000

[12] Warmer, J., Kleppe, A.: The Object Constraint Language. Precise Modeling with UML. Addison-Wesley, 1999

[13] Warmer, J., Kleppe, A.: The Object Constraint Language Second Edition. Getting Your Models Ready For MDA. Addison-Wesley, 2003

[14] Workshop OCL 2.0 - Industry standard or scientific playground?, Sixth International Conference on the Unified Modelling Language - the Language and its applications (UML 2003), October 21, 2003, San Francisco, i11www.ilkd.uni-karlsruhe.de/ baar/oclworkshopUml03/

[15] Zschaler, St.: Evaluation der Praxistauglichkeit von OCL-Spezifikationen. master thesis, Dresden University of Technology, 2002

[16] OMG MOF specification, www.omg.org/technology/documents/formal/mof.htm

[17] OCL 2.0 Submission, www.klasse.nl/ocl/ocl-subm.html

[18] OMG UML v. 1.5 specification, www.omg.org/technology/documents/formal/uml.htm

[19] OMG UML2 Working Documents, www.omg.org/technology/documents/modeling_spec_catalog.htm

[20] OMG, XML Metadata Interchange (XMI). www.omg.org

[21] Mario Jeckle - UML Tools, www.jeckle.de/umltools.htm

[22] Objects by Design list of UML tools, www.objectsbydesign.com/tools/umltools_byCompany.html

[23] GNU Library General Public License, www.gnu.org/copyleft/lgpl.html

[24] Dresden OCL Toolkit, dresden-ocl.sourceforge.net/

[25] Wiebicke, R., Utility Support for Checking OCL Business Rules in Java Programs. master thesis, Dresden University of Technology, 2000, dresden-ocl.sourceforge.net/

[26] ArgoUML tool, argouml.tigris.org/

[27] Poseidon tool, www.gentleware.com/

[28] The KeY Project, www.key-project.org/

[29] OCLE tool, lci.cs.ubbcluj.ro/ocle/

[30] Octopus tool, www.klasse.nl/ocl/octopus-intro.html

[31] Object Constraint Language Library, www.cs.kent.ac.uk/projects/ocl/

[32] USE tool, dustbin.informatik.uni-bremen.de/projects/USE/

[33] Magic Draw UML tool, www.magicdraw.com

[34] Bold for Delphi, info.borland.com/techpubs/delphi/boldfordelphi/

[35] UMLAUT tool, www.irisa.fr/pampa/UMLAUT/

[36] Together tool, www.borland.com/together/

[37] Rational software, www-306.ibm.com/software/rational/

[38] Eclipse platform, www.eclipse.org