Tool Support for OCL and Related Formalisms - Needs and Trends

Thomas Baar[1] and Dan Chiorean[2] and Alexandre Correa[3] and Martin Gogolla[4] and Heinrich Hußmann[5] and Octavian Patrascoiu[6] and Peter H. Schmitt[7] and Jos Warmer[8]

[1] École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
[2] "Babes-Bolyai" University of Cluj-Napoca, Romania
[3] University of Rio de Janeiro, Brazil
[4] University of Bremen, Germany
[5] LMU Munich, Germany
[6] University of Kent, United Kingdom
[7] Universität Karlsruhe, Germany
[8] Ordina, The Netherlands

Abstract. The recent trend in software engineering to model-centered methodologies is an excellent opportunity for OCL to become a widely used specification language. If the focus of the development activities is shifted from implementation code to more abstract models then software developers need a formalism to provide a complete, unambiguous and consistent model at a very detailed level. OCL is currently the only language that can bring this level of detail to UML models. The purpose of the workshop was to identify future challenges for OCL and to discuss how OCL and its current tool support can be improved to meet these challenges. The workshop gathered numerous experts from academia and industry to report on success stories, to formulate wishes to the next generation of OCL tools, and to identify weaknesses in the language, which make OCL sometimes cumbersome to use. The workshop could also attract numerous people whose aim was to get an overview on the state of the art of OCL tool support and on how OCL can efficiently be applied in practice.

1 Motivation and Goals

Model-centric methodologies see modeling artifacts as the primary output of development activities and not implementation code, as it is currently the case in most software development projects. These new methodologies were triggered by recent standardizations of meta-modeling technologies, which have facilitated the syntactic and semantic specification of modeling languages. It has been reported in numerous case studies how model-centric approaches can yield a productivity leap and thus dramatically reduce development costs. Model-centric methodologies could, however, not become mainstream yet, because this would require a matured, seamless tool support for all development phases. One of today's great

challenges is to make modeling tools as powerful and easy to use as current Integrated Development Environments (IDEs) for programming languages.

The Object Constraint Language (OCL) is a standardized, versatile, multi-purpose specification language. It can bring a degree of preciseness to graphical models that is needed if the graphical models should become the *primary* artifacts in the development process. The pressure to improve the tool support for OCL goes along with the overall challenge to improve the quality of modeling tools in general. Improved tool support is just one thing that has to be addressed in order to increase the popularity of OCL. There are plenty of other questions this workshop was devoted to. The following list is a (surely incomplete) classification of questions that need to be answered.

Technical questions on how to improve tool support for OCL There is a technical dimension how the community can effectively provide better OCL tool support. How can we facilitate the development of new tools? Which features should an OCL tool offer to encourage the usage of OCL in practice? Is it feasible to make OCL executable and to provide an animator for OCL? Should we strive for a common architecture of OCL tools which would enable us to reuse standard components, such as a parsing component?
These and similar questions are discussed in the workshop papers [5–8]. The project described in [6] provides a new grammar for OCL that can be used as a starting point to build a parser (actually, the grammar has been already 'implemented' in form of a parser). This is a remarkable step forward since the official grammar given in the OCL 2.0 language specification intentionally abstracts from 'implementation glitches'.

Language issues The language specification for OCL has certainly improved over the last years, but there are still some debatable points in the OCL semantics. Furthermore, OCL is missing some constructs, e.g. a modifies clause, that are widely accepted in the specification language community and that are offered by other specification languages such as the Java Modeling Language (JML).
The paper [9] strives to find a solution for the frame problem that has been highly neglected in OCL so far. The papers [8, 5] discuss besides the semantics of certain constructs also the general architecture of OCL. They try to classify the concepts according to their importance from the language architecture's point of view. Based on this concept classification, the definition of OCL in syntax and semantics could be reorganized to make it more flexible and to define OCL rather as a family of languages than one, entirely fixed language.

Usability questions and application examples Besides improved tool support and a clear and concise language description, OCL would also benefit from more convincing examples and application scenarios.
The paper [3] applies OCL to make the terminology used by meta-modeling experts much more precise. So far, there was a rudimental common agreement among meta-modelers on what basic meta-modeling concepts are supposed to mean but this agreement has never been formalized beforehand.

The paper [4] describes an approach to guide the user when writing OCL constraints. The paper shows how a widely-used class of OCL constraints can actually be generated by instantiating a schematic OCL constraint. This technique is especially suitable for software developers who write their first constraints and want to become familiar with the language.

1.1 Organization

The workshop continued a series of OCL workshops held at previous UML conferences: York, 2000, Toronto 2001, San Francisco, 2003, and Lisbon, 2004. The workshop was organized by the authors of this article, some of them where already involved in the organization of previous OCL workshops and some of them joint the organization team for the first time.

The organizing team formed also the programme committee of the workshop. Each workshop submission received 2-4 reviews written by the members of the organizing team. Based on the reviews, the decision on the paper acceptance was taken unanimously. For papers that were co-authored by one of the workshop organizers, the review process, of course, ensured that the authors had no influence on the acceptance/rejection decision for papers written by themselves.

2 Accepted Papers

All papers accepted at the workshop are published in [1], what can be downloaded either from EPFL's publication archive `http://infoscience.epfl.ch` or from the workshop's website [2]. For the convenience of the reader, we have included here the abstract of each paper.

Title: *On Squeezing M0, M1, M2, and M3 into a Single Object Diagram*
Authors: Martin Gogolla, Jean-Marie Favre, Fabian Büttner
Abstract: We propose an approach for the integrated description of a metamodel and its formal relationship to its models and the model instantiations. The central idea is to use so-called layered graphs permitting to describe type graphs and instance graphs. A type graph can describe a collection of types and their relationships whereas an instance graph can represent instances belonging to the types and respecting the relationships required by the type graph. Type graphs and instance graphs are used iteratively, i.e., an instance graph on one layer can be regarded as a type graph of the next lower layer. Our approach models layered graphs with a UML class diagram, and operations and invariants are formally characterized with OCL and are validated with the USE tool. Metamodeling properties like strictness or well-typedness and features like potency can be formulated as OCL constraints and operations. We are providing easily understandable definitions for several metamodeling notions which are currently used in a loose way by modelers. Such properties and features can then be discussed on a rigorous, formal ground. This issue is also the main purpose of the paper, namely, to provide a basis for discussing metamodeling topics.

Title: *Formal Description of OCL Specification Patterns for Behavioral Specification of Software Components*
Author: Jörg Ackermann
Abstract: The Object Constraint Language (OCL) is often used for behavioral specification of software components. One current problem in specifying behavioral aspects comes from the fact that editing OCL constraints manually is time consuming and error-prone. To simplify constraint definition we propose to use specification patterns for which OCL constraints can be generated automatically. In this paper we outline this solution proposal and develop a way how to formally describe such specification patterns on which a library of reusable OCL specifications is based.

Title: *Supporting OCL as part of a Family of Languages*
Authors: David H. Akehurst, Gareth Howells, Klaus D. McDonald-Maier
Abstract: With the continued interest in Model Driven techniques for software development more and more uses are found for query or expression languages that navigate and manipulate object-oriented models. The Object Constraint Language is one of the most frequently used languages; however, its original intended use as a constraint expression language has been succeeded by its frequently proposed use as a basis for a more general model query language, model transformation language and potential action language. We see a future where OCL forms a basis for a family of languages related in particular to Model Driven Development techniques; as a consequence we require an appropriate tool suite to aid in the development of such language families. This paper proposes some important aspects of such a tool suit.

Title: *Generation of an OCL 2.0 Parser*
Authors: Birgit Demuth, Heinrich Hussmann, Ansgar Konermann
Abstract: The OCL 2.0 specification defines explicitly a concrete and an abstract syntax. The concrete syntax allows modelers to write down OCL expressions in a textual way. The abstract syntax represents the concepts of OCL using a MOF compliant metamodel. OCL 2.0 implementations should follow this specification. In doing so emphasis is placed on the fact that at the end of the processing a tool should produce the same well-formed instance of the abstract syntax as given in the specification. This offers the possibility to implement OCL-like languages with the same semantics that are for example easier to use for business modelers. Therefore we looked for a parser technique that helps us to generate an OCL parser to a large extent. In this paper we present the technique we developed and proved within the scope of the Dresden OCL Toolkit. The resulting Dresden OCL2 parser is especially characterized by using a generation approach not only based on a context-free grammar but on an attribute grammar to create the required instance of the abstract syntax of an OCL expression.

Title: *Lessons Learned from Developing a Dynamic OCL Constraint Enforcement Tool for Java*
Authors: Wojciech J. Dzidek, Lionel C. Briand, Yvan Labiche

Abstract: Analysis and design by contract allows the definition of a formal agreement between a class and its clients, expressing each partys rights and obligations. Contracts written in the Object Constraint Language (OCL) are known to be a useful technique to specify the precondition and postcondition of operations and class invariants in a UML context, making the definition of object-oriented analysis or design elements more precise while also helping in testing and debugging. In this article, we report on the experiences with the development of ocl2j, a tool that automatically instruments OCL constraints in Java programs using aspect-oriented programming (AOP). The approach strives for automatic and efficient generation of contract code, and a non-intrusive instrumentation technique. A summary of our approach is given along with the results of an initial case study, the discussion of encountered problems, and the necessary future work to resolve the encountered issues.

Title: *Proposals for a Widespread Use of OCL*
Authors: Dan Chiorean, Maria Bortes, Dyan Corutiu
Abstract: In spite of the fact that OCL and UML evolved simultaneously, the usage of the constraint language in modeling real-world applications has been insignificant compared to the usage of the graphical language. Presently, OCL is requested in new modeling approaches: Model Driven Architecture, Model Driven Development, Domain Specific Languages, Aspect Oriented Modeling, and various emerging technologies: Semantic Web, Business Rules. In this context, the question What has to be done for OCL to become the rule, not the exception, in the modeling domain? is more pressing than ever. The purpose of this paper is to propose an answer to this question, although not a complete one. Our work is an attempt to synchronize the language specification and its understanding, straight related to the language implementation in CASE tools, by proposing solutions for incomplete or non-deterministic OCL specifications. In order to manage the new extensions required for the constraint language, a new language structure is suggested.

Title: *OCL and Graph Transformations – A Symbiotic Alliance to Alleviate the Frame Problem*
Author: Thomas Baar
Abstract: Many popular methodologies are influenced by Design by Contract. They recommend to specify the intended behavior of operations in an early phase of the software development life cycle. In practice, software developers use most often natural language to describe how the state of the system is supposed to change when the operation is executed. Formal contract specification languages are still rarely used because their semantics often mismatch the needs of software developers. Restrictive specification languages usually suffer from the frame problem: It is hard to express which parts of the system state should remain unaffected when the specified operation is executed. Constructive specification languages, instead, suffer from the tendency to make specifications deterministic.
This paper investigates how a combination of OCL and graph transformations can overcome the frame problem and can make constructive specifications less

deterministic. Our new contract specification language is considerably more expressive than both pure OCL and pure graph transformations.

3 Workshop Results

The workshop attracted 38 registered attendees. The motivation was rather divers; some of them wanted to learn OCL and to get acquainted with it, others came to discuss specific problems in depth.

3.1 Spontaneous Tool Overview Session

Since the workshop attracted many people from academia and industry who were not OCL experts but wanted to get an overview on OCL technology, Martin Gogolla made, after every workshop participant has introduced himself, the suggestion to devote the first session to present briefly some of the currently existing OCL tools. This was especially attractive since some of the tool developers were sitting in the workshop room. The following tools were informally presented to the audience:

Martin Gogolla: USE tool The USE tool (UML-based Specification Environment)[9] supports analysts, designers and developers in executing UML models and checking OCL constraints and thus enables them to employ model-driven techniques for software production. USE allows the validation of UML models and OCL constraints based on animation and certification. USE permits analyzing the model structure (classes, associations, attributes, and invariants) and the model behavior (operations and pre- and postconditions) by generating typical snapshots (system states) and by executing typical operation sequences (scenarios). Developers can formally check constraints (invariants and pre- and postconditions) against their expectations and can, to a certain extent, derive formal model properties.

Heinrich Hußmann: Dresden OCL2 Toolkit The Dresden OCL2 Toolkit[10] is a set of tools for processing OCL specifications. The heart of the toolkit is a recently redesigned parser for OCL 2.0. The Abstract Syntax Tree (AST) produced by the parser conforms with the official metamodel of OCL 2.0. Other tools in the toolkit can translate OCL specifications into SQL queries or into Java code, which is able to check the correctness of OCL assertions at runtime.

Behzad Bordbar: UML2Alloy UML2Alloy[11] allows to translate UML class diagrams enriched with OCL expressions into models written in Alloy. Class diagrams are used to depict the static structure of the system. OCL statements are used to both define behavior through pre- and postconditions, and invariants on the UML class diagrams. The tool accepts a UML model

[9] See `http://www.db.informatik.uni-bremen.de/projects/USE/`
[10] See `http://dresden-ocl.sourceforge.net`
[11] See `http://www.cs.bham.ac.uk/~bxb/UML2Alloy.html`

of the system in XMI format and guides the user step by step through the translation of the UML model to a corresponding Alloy model. Users can then use Alloy Analyzer on the produced Alloy model to conduct analysis. Alloy Analyzer provides the ability of Analysis. This includes simulation of the system, which provides examples of instances that conform to the model. This is particularly helpful in checking if the model is overconstrained and to increase the confidence in correctness of the model. It is also possible to check the correctness of logical statements, assertions, about the model.

Thomas Baar: KeY tool and Grammatical Framework (GF) The KeY tool[12] is not primarily an OCL tool but has an OCL front end (in fact, it uses the OCL parser from the Dresden OCL toolkit) for behavioral specification of operations in a UML class diagram. The KeY tool allows the user to verify the correctness of operation implementations written in Java in respect to an OCL specification given as a pair of pre- and postcondition (contract). Otherwise stated, using KeY one can statically prove that whenever the operation's implementation is invoked in a state in which the pre-condition holds, the execution of the implementation will terminate and yield to a state in which the postcondition holds.

The Grammatical Framework (GF) is designed as a stand-alone tool but has been fully integrated into the KeY tool. The Grammatical Framework offers translations of OCL specifications into natural language. Most developers appreciate if OCL constraints are presented in natural language since – as for any other formal specification language – it is time consuming to read and to understand formal OCL constraints. Languages, currently supported by GF as a target language, are English, Swedish, Finnish, German. Also the opposite direction of translation, from natural language to OCL, is prototypically realized.

3.2 Discussion

The last session of the workshop was devoted to discussion on OCL issues raised during the paper presentation sessions as well as other issues that are of common interest. The following list captures the main points of the discussion. Many problems remained unsolved and it was not always possible to come to an agreement among all participants.

OCL must be supported by better tools.

Most of the current OCL tools are academic tools and were developed by a team of a single university. Although the quality of tools has improved considerably over the last years, it is not a surprise that these OCL tools cannot compete in terms of usability and the functionality they offer with integrated development environments for writing implementation code.

One trap a lot of OCL tool development teams fall into is to capture every possible application scenario for OCL by their tool. Instead of a one-fits-it-all-tool we need rather a component-oriented approach where specialized

[12] See `http://www.key-project.org`

tools provide services using standardized interfaces and other tools can take advantages of them. Examples for such services could be: parse a constraint, evaluate a constraint, pretty print a constraint, find counterexample that constraint always holds, generate implementation code, etc. A first step towards this goal could be to define a list of functionalities a user would expect from a matured OCL tool. The list should also clarify in which scenario the functionality would be useful.

Applying OCL yields to better software and saves valuable time.
Based on the current examples and case studies it is hard to convince software developers on the usefulness of applying OCL in practice. There are even experiences reported in the literature where an OCL specification of a Java framework is considered to be less informative than other ways to specify the framework, e.g. by a reference implementation or carefully written informal comments. On the other hand, a few controlled experiments conducted in academic settings have reported positive results on using OCL in UML based developments.

Developers might be convinced more easily once we had compelling results from more experiments available. For instance, it would be interesting to set up two teams developing the same application in parallel and measure the effort and the quality of the resulting artifacts. One team uses OCL assertions whereas the other tries to model and implement the application the traditional way without OCL assertions. Such an experiment can hardly be done in real software industry but it is possible to run it at universities with two groups of students (trained in OCL and without any knowledge on OCL).

Promising application areas for OCL have to be identified.
We need a clear idea on what are the most promising application scenarios for OCL. If OCL is used at the very detailed level of implementation models to describe the behavior of implemented methods, then it competes for instance with JML. In this case, OCL is often not chosen as the specification language because it's semantics is not aligned enough with this application area. For instance, the type system of OCL refers to that of UML and do not take the peculiarities of Java's type system into account (however, the Java type system could be made available to OCL via a Java profile).

But weaknesses on one side are strengths on the other side. Since OCL is fully integrated into the UML metamodel, it can specify properties directly at any level of abstraction. As another advantage, OCL provides powerful mechanisms for reflection and allows the user to explore the metamodel within a constraint.

OCL is a family of languages.
The application scenarios of OCL are very divers and require sometimes to adapt the semantics of certain constructs to the current scenario or to add new, scenario-specific constructs. This gives rise to treat OCL as a family of languages instead of a fixed one. The OCL language specification should be

rewritten according to this fact and should allow the user to customize the currently needed dialect of OCL. The possibility to customize the language has of course to be backed by the tools that support OCL. Either a tool can be customized by the user, i.e. the tool is tailored to the OCL dialect the user has in mind, or the tool clearly states which of OCL's dialects it supports.

More teaching modules on the art of specification are needed.
There is still a lack of good teaching modules for OCL and the diversity among the illustrating examples for OCL constraints is rather low. Also case studies on bigger projects would help many potential users to find out whether or not OCL is the proper formalism to describe the problems they have.
It was decided on the workshop to launch a new website as an archive of existing teaching modules, experience reports, etc. This website is already available under `http://www-st.inf.tu-dresden.de/ocl/`. Everybody is encouraged to contribute!

Acknowledgement

The authors are grateful to Behzad Bordbar and Dave Akehurst for their comments on earlier drafts of this workshop report.

References

1. Thomas Baar, editor. *Tool Support for OCL and Related Formalisms - Needs and Trends, MoDELS'05 Conference Workshop, Montego Bay, Jamaica, October 4, 2005, Proceedings*, Technical Report LGL-REPORT-2005-001. EPFL, 2005.
2. Homepage of OCL Workshop 2005. `http://lgl.epfl.ch/members/baar/oclws05`.
3. Martin Gogolla, Jean-Marie Favre, and Fabian Büttner. On squeezing M0, M1, M2, and M3 into a single object diagram.
4. Jörg Ackermann. Formal description of OCL specification patterns for behavioral specification of software components.
5. David H. Akehurst, Gareth Howells, and Klaus D. McDonald-Maier. Supporting OCL as part of a family of languages.
6. Birgit Demuth, Heinrich Hussmann, and Ansgar Konermann. Generation of an OCL 2.0 parser.
7. Wojciech J. Dzidek, Lionel C. Briand, and Yvan Labiche. Lessons learned from developing a dynamic OCL constraint enforcement tool for Java.
8. Dan Chiorean, Maria Bortes, and Dyan Corutiu. Proposals for a widespread use of OCL.
9. Thomas Baar. OCL and graph transformations – a symbiotic alliance to alleviate the frame problem.