International Workshop on Ocl4All (2007): Modelling Systems with OCL

David Akehurst[1], Martin Gogolla[2], and Steffen Zschaler[3]

[1] Thales Scotland
dave@akehurst.net
[2] Technische Universität Bremen
gogolla@informatik.uni-bremen.de
[3] Technische Universität Dresden
szschaler@acm.org

Abstract. This year's OCL workshop at the MODELS conference looked out to usages of OCL outside the direct context of UML or beyond the capabilities of standard OCL. It was a very interesting and successful workshop, which apart from the presentation of 10 papers a lively discussion on various topics surround current usage of OCL was held. Six main topics recurred throughout the discussions: (1) Means and uses of transformation to other languages, (2) Support for side-effects and executability, (3) Continuing need for OCL, (4) Providing extensions to the standard, (5) Platform independence of the language, (6) Formality of the language. This summary report presents the results of the workshop and the discussions.

1 Introduction

Ocl4All (2007) is the seventh UML/MODELS workshop on the Object Constraint Language (OCL). As with its previous editions, this year's OCL workshop has been held under a specific theme: Using OCL in an extended system-modelling context. Since the inception of OCL, the world of modelling has changed. While the Unified Modelling Language (UML) is still of great importance, many software-development projects use other, often so-called domain-specific languages for their models. In this situation, OCL must go beyond its tight connection to UML or it must eventually fail. And, indeed, we see a strong movement for extensions of OCL that bring it beyond its original concepts both in its relation to UML and in the basic principles underlying the language. At the same time, the current OCL 2.0 standard already spans some 230 pages making it quite hard to ensure consistency both in the standard and in its implementations. Therefore, the OCL community needs to start looking for ways of systematically modularising and extending the language. The 2007 issue of the OCL workshop aimed to discuss cases of using OCL outside of UML, problems encountered and solutions proposed. The workshop was very lively, in a mixture of presentations and discussions. Out of a submitted 14 papers, 10 were accepted. The accepted papers have been published in a special edition of Electronic Communications of the EASST, an on-line journal [1]. In this workshop summary, we first present a short overview of the papers and then go on to the main points of discussion and the conclusions reached at the workshop.

2 Workshop Papers

Here, we give a short overview for each paper taken from the abstracts provided by the authors.

2.1 William Robinson: Extended OCL for Goal Monitoring [2]

Monitoring human-computer interaction aids the analysis for understanding how well software meets its purpose. In particular, monitoring human-computer interactions with respect to a users goal model helps to determine user satisfaction. By formalizing a goal model, runtime monitors can be automatically derived. The REQMON system monitors the satisfaction of goal models. Recently, an OCL compiler was developed for REQMON. The OCL was extended slightly to address temporal and real-time constraints. Now, goal models can be represented in the extended OCL, from which runtime monitors can be compiled. The resulting REQMON system appears to be easier to use.

2.2 Pierre Kelsen et al.: Specifying Executable Platform-Independent Models using OCL [3]

Model-driven architecture aims at describing a system using a platform-independent model in sufficient detail so that the full implementation of the system can be generated from this model and a platform model. This implies that the platform-independent model must describe the static structure as well as the dynamic behavior of the system. We propose a declarative language for describing the behavior of platform-independent models based on a hybrid notation that uses graphical elements as well as textual elements in the form of OCL code snippets. Compared to existing approaches based on action languages it is situated at a higher level of abstraction and, through a clean separation of modifier operations and query operations, simplifies the comprehension of the behavioral aspects of the platform-independent system.

2.3 Ke Jiang et al.: Using OCL in Executable UML [4]

Executable UML allows precisely describing the software system at a higher level of abstraction. The executable models can be translated to a less abstract programming language completely or executed directly. Object Constraint Language (OCL), as a formal specification language, is a standard published along with UML. It is primitively used to describe constraints for UML models. In this paper, we explore some general features of executable UML and propose using OCL in executable UML. We extend OCL to support actions with side-effect in order to precisely model behavior. We also discuss some open issues.

2.4 David Akehurst et al.: OCL: Modularising the Language [5]

The Object Constraint Language (OCL) was originally designed as an 'add-on' to the Unified Modelling Language (UML) in order to facilitate writing textual constraints complementing the graphical specifications. Since its original standardisation many extensions have been added to the language and many more have been proposed. The original structure of the OCL definition has not been formed, however, with a view of extensibility. Still, OCL can be redesigned in such a manner that it becomes easy to extend the language. In this paper we present a modular redefinition of OCL and illustrate how it supports extension. This new approach to the design of OCL enables you to consistently extend or customise OCL to your own needs.

2.5 Mirco Kuhlmann et al.: Analyzing Semantic Properties of OCL Operations by Uncovering Interoperational Relationships [6]

The OCL (Object Constraint Language) as part of the UML (Unified Modeling Language) is a rich language with different collection kinds (sets, multi-sets, sequences) and a large variety of operations defined thereon. Without negating the strong correlation between both fields we can say that these operations have their origin partly in logic (like the operations forAll and exists) and partly in computer science, in particular database systems (like the operation select). Some of these operations may be expressed in terms of other operations. This paper presents a systematic study of relationships which hold between OCL features like the mentioned operations. Apart from presenting the relationships between operations in a conceptual way, the relationships are described by a formal metamodel allowing systematic and computer supported access to the operation relationships by querying an underlying formal description.

2.6 David Akehurst et al.: C# 3.0 makes OCL redundant! [7]

Other than its 'platform independence' the major advantages of OCL over traditional Object Oriented programming languages has been the declarative nature of the language, its powerful navigation facility via the iteration operations, and the availability of tuples as a first class concept. The recent offering from Microsoft of the "Orcas" version of Visual Studio with C# 3.0 and the Linq library provides functionality almost identical to that of OCL. This paper examines and evaluates the controversial thesis that, as a result of C# 3.0, OCL is essentially redundant, having been superseded by the incorporation of its advantageous features into a mainstream programming language.

2.7 Milan Milanović et al.: Sharing OCL Constraints by Using Web Rules [8]

This paper presents an MDE-based approach to interchanging rules between the Object Constraint Language (OCL) and REWERSE I1 Rule Markup Language

(R2ML). The R2ML tends to be a standard rule markup language by following up the W3C initiative for Rule Interchange Format (RIF). The main benefit of this approach is that the transformations between languages are completely based on the languages' abstract syntax (i.e., metamodels) and in this way we keep the focus on the language concepts rather than on technical issues caused by different concrete syntax. In the current implementation, we have supported translation of the OCL invariants into the R2ML integrity rules. While most of the OCL expression could be represented in the R2ML and other rule languages, we have also identified that collection operators could only be partially supported in other rule languages (e.g., SWRL).

2.8 Florian Heidenreich et al.: A Framework for Generating Query Language Code from OCL Invariants [9]

The semantical integrity of business data is of great importance for the implementation of business applications. Model-Driven Software Development (MDSD) allows for specifying the relevant domain concepts, their interrelations and their concise semantics using a plethora of modelling languages. Since model transformations enable an automatic mapping of platform independent models (PIMs) to platform specific models (PSMs) and code, it is reasonable to utilise them to derive data schemas and integrity rules for business applications. Most current approaches only focus on transforming structural descriptions of software systems while semantical specifications are neglected. However, to preserve also the semantical integrity rules we propose a Query Code Generation Framework that enables Model-Driven Integrity Engineering. This framework allows for mapping UML models to arbitrary data schemas and for mapping OCL invariants to sentences in corresponding declarative query languages, enforcing semantical data integrity on implementation level. This supersedes the manual translation of integrity constraints and, thus, decreases development costs while increasing software quality.

2.9 Matthias Bräurer et al.: Model-Level Integration of the OCL Standard Library Using a Pivot Model with Generics Support [10]

OCL 2.0 specifies a standard library of predefined types and associated operations. A model-level representation of the library is required to reference its elements within the abstract syntax model created by an OCL parser. Existing OCL engines build this model in the implementation code which severely limits reusability, flexibility and maintainability. To address these problems, we show how a common pivot model with explicit support for template types can help to externalize the definition of the standard library and integrate it with instances of arbitrary domain-specific modeling languages. We exemplify the feasibility of our approach with a prototypical implementation for the Dresden OCL2 Toolkit and present a tailored EMF editor for modeling the OCL types and operations.

We limit our discussion to the model level, i.e., we do not consider an implementation of the standard library for an execution engine.

2.10 Emine Aydal: Evaluation of OCL for Large-Scale Modelling: A Different View of the Mondex Smart Card Application [11]

OCL is used to add rigour to UML/MOF models, and in particular can be used to express behavioural details (e.g., operation pre- and postconditions, class invariants) of such models. The applicability and utility of OCL can be assessed by applying it to realistic applications and by investigating its capabilities both in terms of language characteristics and tool support. With this in mind, in this paper we model functional requirements for the Mondex Smart Card Application using UML Diagrams, demonstrate how system invariants as well as operation pre- and post-conditions are specified in OCL, and explore the degree to which OCL tool support can be used to create and validate these models. Moreover, we discuss how these pre- and post-conditions can be validated, in part by discussing how test cases can be selected from the OCL specifications created.

3 Workshop Discussions

There were a number of recurring topics that occurred throughout the presentations and discussion of the workshop: (1) Transformation to other languages, (2) Support for side-effects and executability, (3) Is OCL redundant, (4) Providing extensions to the standard, (5) Platform independence, (6) Is it a formal language?

The transformation of OCL to other languages was directly raised in two papers [8, 9] regarding the conversion to web rule languages and to SQL. Open world vs closed world assumptions was a distinguishing point between these two papers. The SQL paper stayed in a semantics with closed world assumption whilst the web rule paper switched to open world. OCL uses a closed world assumption therefore translating to an open world language causes problems in particular with negations.

There was extensive discussion on whether OCL should support side-effects. This related to other discussions on making OCL more easily extensible so that side-effect concepts (amongst others) can be added if required. There was clear agreement that OCL is currently intended to be side-effect free. The discussion took two directions, one regarding whether or not it would be useful and the other on potential problems it would cause. There was no agreement on whether it would be useful; the main argument for side-effect was the goal of full code generation, however the counter argument revolved around OCL being declarative and not focusing on implementation. Potential problems were discussed in the areas of object identity, query operations and local let expressions.

Triggered by paper [7], discussion occurred on the potential redundancy of OCL in the context of other programming languages such as C# and Ruby, which support some of the same features. The main two arguments in favour of OCL were the potential for verification and validation support and the small size of OCL.

Many papers discussed extensions to OCL [2–4] and there was common agreement that a standard mechanism for enabling these extensions to be clearly and easily defined was required. This is related to other discussions and papers on redefining the OCL standard in a more modular fashion. There were two papers that discussed the requirements for the OCL standard library to be replaceable [5, 10].

The question of whether OCL is platform independent was discussed. This related to discussion on translating OCL to other languages and issues about the standard library. No definitive conclusion was reached.

Finally a long discussion was held on whether OCL is a formal language. The discussion mainly revolved around the provision of a formal semantics for OCL. Past work has provided a formal semantics for parts of and different versions of OCL. The current standard certainly does not provide a full formal semantics. Thus we agreed that OCL is a semi-formal language as it is currently defined.

References

1. Akehurst, D.H., Gogolla, M., Zschaler, S., eds.: Ocl4All: Modelling Systems with OCL. Volume 9 of Electronic Communications of the EASST. (2008) Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
2. Robinson, W.: Extended OCL for goal monitoring. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
3. Kelsen, P., Pulvermueller, E., Glodt, C.: Specifying executable platform-independent models using OCL. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
4. Jiang, K., Zhang, L., Miyake, S.: Using OCL in executable UML. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
5. Akehurst, D.H., Zschaler, S., Howells, G.: OCL: Modularising the language. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
6. Kuhlmann, M., Gogolla, M.: Analyzing semantic properties of OCL operations by uncovering interoperational relationships. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
7. Akehurst, D.H., Howells, G., Scheidgen, M., McDonald-Maier, K.: C# 3.0 makes OCL redundant! [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
8. Milanović, M., Gašević, D., Giurca, A., Wagner, G., Devedžić, V.: Sharing OCL constraints by using web rules. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
9. Heidenreich, F., Wende, C., Demuth, B.: A framework for generating query language code from OCL invariants. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
10. Bräuer, M., Demuth, B.: Model-level integration of the OCL standard library using a pivot model with generics support. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.
11. Aydal, E.G., Paige, R.F., Woodcock, J.: Evaluation of OCL for large-scale modelling: A different view of the Mondex purse. [1] Available from http://eceasst.cs.tu-berlin.de/index.php/eceasst/issue/view/16.