Workshop in OCL and Textual Modelling
Report on Recent Trends and Panel Discussions

Robert Bill[1], Achim D. Brucker[2], Jordi Cabot[3,4], Martin Gogolla[5], Antonio Vallecillo[6], and Edward D. Willink[7]

[1] TU Vienna, Austria
bill@big.tuwien.ac.at
[2] The University of Sheffield, Sheffield, UK
a.brucker@sheffield.ac.uk
[3] ICREA, Spain
[4] UOC, Spain
jordi.cabot@icrea.cat
[5] University of Bremen, Bremen, Germany
gogolla@informatik.uni-bremen.de
[6] Universidad de Málaga, Málaga, Spain
av@lcc.uma.es
[7] Willink Transformations Ltd, Reading, UK
ed_at_willink.me.uk

Abstract This paper reports on the panel session of the 17th Workshop in OCL and Textual Modeling, As in previous years, the panel session featured several lightning talks for presenting recent developments and open questions in the area of OCL and textual modeling. During this session, the OCL community discussed, stimulated through short presentations by OCL experts, proposals for improving OCL to increase the attractiveness of textual modeling.
This paper contains a summary of the workshop from the workshop organisers as well as summaries of two lightning talks provided by their presenters.

Keywords: OCL · textual modeling

1 Introduction

Textual modeling in general and OCL in particular are well established. This year does not only mark the 17th edition of the OCL workshop, it also marks the twentieth anniversary of the first publication of the OCL standard by the OMG [3]. Nevertheless, textual modeling in general and OCL in particular is an active field of research.

The workshop received seven submissions from which five were selected as full papers. Each paper was reviewed by at least three PC members. The workshop hosted an open session with "Lightning Talks (5 minutes)" at the end of the day where speakers were given the opportunity to talk about whatever they wanted,

M. Seidl and S. Zschaller (Eds.): STAF Workshop Proceedings, LNCS 10748, pp. 280–284, 2018.

as long as it was related to the topics of the workshop. Three presentations were given. The topics discussed at the workshop covered topics such as the translation of OCL to programming and specification languages, proposals for improving textual modeling languages and their tool support, as well as the development of an OCL benchmark.

The lighting talks at the panel session of the workshop provided a platform for the textual modeling community to discuss and present tools, ideas, and proposals to support textual modeling as well as to shape the future of textual modeling. The following sections, each of them contributed by one expert of the field, discuss the different tools and ideas that were discussed during the panel session.[8]

2 Sometimes Postconditions Do Not Suffice

Martin Gogolla and Antonio Vallecillo

2.1 Non-Determinateness and Randomness in OCL

Recently there have been proposals for incorporating the option to express randomness in OCL [2, 4]. In many modeling and simulation environments, the use of random numbers and probability distributions are used to combine definite knowledge with an uncertain view on the result or the population of a test case. Thus, there is an interest to express such requirements in UML and OCL.

OCL already has operations that possess a flavor of randomness, like the operation `any()`. One could also consider a new collection operation `random()` that randomly chooses an element from the argument collection. Our understanding of such operations is that they cannot be characterized only by 'traditional' postconditions. In particular special attention has to be given in order to express the difference between `any()` and `random()`: A 'traditional' postcondition would characterize 'one' call to the respective operation (for example, with `Set{1..6}->includes(result)`); but these two operations must be characterized by 'many' operation calls and a comparison between their actual and their expected results. We show with a small example how such a 'non-traditional' postcondition in form of an invariant could look like.

2.2 Formulating Randomness Quality Criteria as an Invariant

Consider the class diagram in Fig. 1 that is intended to model a dice. Every time the operation `random6()` is called it should return a random number between `1` and `6`. Our expectation for the operation `any()` would be that it can also return any number between `1` and `6`, but that different calls to `any()` always yield the same result. In contrast, different calls to `random()` should show different results.

[8] There is no summary for the lightning talk of Dimitris Kolovos, entitled "Managing MATLAB Simulink models with Epsilon," as the author considered the results to be in an too early stage to be summarized in a written report.

The attributes in the class Dice (see Lst. 1.1) give a simple measure for the quality of the generated random numbers. Basically the attributes say that the number of tests for random6() that have to be performed is numChecks and that, for example, the difference between (a) the amount of operation calls yielding 2 and (b) the amount of operation calls yielding 5 is at most deltaMax. These requirements are formulated as an OCL formula in terms of an invariant of the class Dice. The requirement should not be formulated as a random6() postcondition because this would lead to a situation where a recursive call to the operation would occur in the postcondition. Much better criteria for the random distribution could be formulated in OCL as well. The purpose of the shown invariant is only to demonstrate that many calls to an operation may be necessary in order to express desired properties.

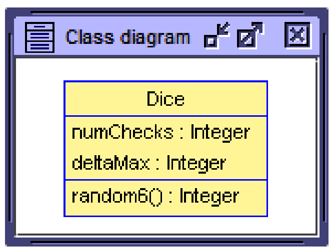

Figure 1. Class diagram for Dice example.

3 Commutative Short Circuit Operators

Edward D. Willink

OCL's 4-level logic has been a source of much unhappiness and while various solutions have been suggested, none have met with enthusiasm. We look at where the unhappiness comes from and thereby suggest a new solution.

The OCL designers defined an underlying model in which all expressions have types. Consequently the mathematical concept of truth was reified by a Boolean type with associated Boolean library operations. The designers chose to avoid exceptions. This in combination with UML conformance required a null value for the missing value of properties with optional multiplicity, and an invalid value for everything bad that might be evaluated.

Unfortunately null and invalid pollute the simplicity of truths and so the Amsterdam Manifesto [1] elaborates Boolean operators with short-circuit like functionality for problems such as:

```
a <> null and a.doSomething()
```

However the operators remain commutative and so it is suggested that all terms are evaluated in parallel until the result is knowable. A Karnaugh Map

```
class Dice
attributes
  numChecks:Integer
  deltaMax:Integer
operations
  random6():Integer=Set{1..6}->random()
  post returns_1_6: Set{1..6}->includes(result)
constraints
  inv manyRandom6CallsResultInNearlyEquallyDistributedValues:
    -- call random6() many times
    -- store resulting amounts in Sequence{A1,A2,A3,A4,A5,A6}
    -- check differences between A1..A6
    let amts=Set{1..numChecks}->iterate(i:Integer;
      amts:Sequence(Integer)=Sequence{0,0,0,0,0,0} |
      let r=random6() in
      Sequence{
        if r=1 then amts->at(1)+1 else amts->at(1) endif,
        if r=2 then amts->at(2)+1 else amts->at(2) endif,
        if r=3 then amts->at(3)+1 else amts->at(3) endif,
        if r=4 then amts->at(4)+1 else amts->at(4) endif,
        if r=5 then amts->at(5)+1 else amts->at(5) endif,
        if r=6 then amts->at(6)+1 else amts->at(6) endif}) in
    Sequence{1..5}->iterate(i; diffs:Sequence(Integer)=Sequence{} |
      Sequence{i+1..6}->iterate(j; diffs2:Sequence(Integer)=diffs |
        diffs2->including((amts->at(i)-amts->at(j)).abs())))->
          forAll(d | d<=deltaMax)
end
```

Listing 1.1. Specification of the Dice example.

defines the mapping from the `true` (T), `false` (F), `null` (ϵ) and `invalid` ($\perp$) values of `Left` and `Right` inputs to the and output.

Left	Right	and	requires	'and2'
T	T	T	T	T
T	F	F	F	F
T	$\perp$,ϵ	$\perp$	$\perp$	$\perp$
F	-		F	
F	T,F	F		F
F	$\perp$,ϵ	F		$\perp$
$\perp$,ϵ	-		$\perp$	
$\perp$,ϵ	T,F,$\perp$,ϵ	$\perp$		$\perp$

Parallel execution is an implementation nightmare and the intermediate `invalid` results can be inefficient. If we eliminate commutative short circuits, we find that `invalid` results are exceptional rather than normal.

```
a <> null requires a.doSomething()
```

A new `requires` operator imposes a left argument first evaluation order for and. This avoids the spurious `invalid` results from the right argument and clearly indicates the intent to handle non-truths. The and operator can then be used for truths only. Once static analysis verifies that neither left nor right input of an and operator can be `null` or `invalid`, an implementation may implement a regular `'and2'` operation that returns `invalid` for any `null` or `invalid` input.

A new `obviates` operator is also needed to regularize `or` short circuiting.

4 Conclusion

The lively discussions both during the lighting talks as well as for each paper that was presented showed again that the OCL community is a very active community. Moreover, it showed that OCL, even though it is a mature language that is widely used, has still areas in which the language can be improved. We all will look forward to upcoming version of the OCL standard and next year's edition of the OCL workshop.

Acknowledgments. We would like to thank all participants of this years OCL workshop for their active contributions to the discussions at the workshop. These lively discussions are a significant contribution to the success of the OCL workshop series.

Bibliography

[1] Cook, S., Kleppe, A., Mitchell, R., Rumpe, B., Warmer, J., Wills, A.: The amsterdam manifesto on OCL. In: Clark, T., Warmer, J. (eds.) Object Modeling with the OCL: The Rationale behind the Object Constraint Language, *Lecture Notes in Computer Science*, vol. 2263, pp. 115–149. Springer-Verlag, Heidelberg (2002)

[2] Johnson, P., Ullberg, J., Buschle, M., Franke, U., Shahzad, K.: P2AMF: Predictive, Probabilistic Architecture Modeling Framework. In: van Sinderen, M., Luttighuis, P.O., Folmer, E., Bosems, S. (eds.) Enterprise Interoperability - Proc. 5th Int. IFIP Working Conf., IWEI, 2013, *LNBIP*, vol. 144, pp. 104–117. Springer (2013)

[3] OMG: Object constraint language specification (version 1.1) (1997). Available as OMG document ad/97-08-08

[4] Vallecillo, A., Gogolla, M.: Adding Random Operations to OCL. In: Posse, E., Ratiu, D., Selim, G., Zalila, F. (eds.) Proc. Workshop on Model Driven Engineering, Verification and Validation (MODEVVA 2017). CEUR Proceedings (2017)