Electronic Communications of the EASST
Volume 44 (2011)

Proceedings of the
Workshop on OCL and Textual Modelling
(OCL 2011)

UML is still inconsistent! How to improve OCL Constraints in the UML 2.3 Superstructure

Claas Wilke and Birgit Demuth

19 pages

Guest Editors: Jordi Cabot, Robert Clariso, Martin Gogolla, Burkhart Wolff
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

UML is still inconsistent! How to improve OCL Constraints in the UML 2.3 Superstructure

Claas Wilke[1] **and Birgit Demuth**[2]

[1]claas.wilke@tu-dresden.de
[2]birgit.demuth@tu-dresden.de
Institut für Software- und Multimediatechnik
Technische Universität Dresden
D-01062, Dresden, Germany

Abstract: Since the first OMG specification of the Unified Modeling Language (UML), the Object Constraint Language (OCL) has been used for the definition of well-formedness rules in the UML specification. These rules have been specified within the early OCL years, when no appropriate tooling existed. Thus, they could not be checked for syntactical and static semantics correctness. In this paper we present an analysis of the static correctness of all OCL rules specified in the UML 2.3 superstructure document. We categorise found errors and propose changes for both the UML specification process and the OCL language to improve the UML specification's correctness in future versions.

Keywords: UML Specification, OCL, Well-Formedness Rules, Consistency Study.

1 Introduction

Since the first Unified Modeling Language (UML) adopted specification was published by the Object Management Group (OMG) in 1997 and formally released as UML 1.3 in 2000, the UML has been revised and extended multiple times [Obj11e]. From the beginning of UML, the Object Constraint Language (OCL) [Obj11b] has been used to specify well-formedness rules (WFRs) of the UML metamodel, and has become a standard constraint language for the WFR definition of other OMG specifications as well [Obj03, Obj06, Obj11a, Obj11c]. When the first OCL rules were defined for the UML metamodel, no OCL tooling was available to parse these rules and hence, no syntax nor semantic checks on these rules could be performed. Although many OCL tools have been developed to date [HDF02, GBR07, MDT11], the WFRs of the UML specification have never been revised nor updated according to changes performed within the UML metamodel and the OCL's concrete syntax and semantics.

In this paper we investigate the static correctness of all OCL constraints defined in the UML 2.3 specification [Obj10] and evaluate and categorise errors found within these constraints. We further derive necessary improvements of the OMG's UML specification process which is—according to our results—still non-model-driven. Finally, we conclude our work by proposing improvements of the OCL itself to avoid similar errors in and ease the specification of future WFRs.

The rest of this paper is structured as follows. In Section 2 we present some related work that has already been done on WFR specifications of the UML. Following in Section 3, the methodology we used to parse and evaluate the OCL constraints is presented. Section 4 highlights the results of our investigation. In Section 5 we discuss these results and propose changes to both the UML's specification process and the OCL. Finally, we conclude our work in Section 6.

2 Related Work

Although acceptable and usable OCL tools exist for more than ten years, only rare work has been done regarding the correctness and consistency of the OCL rules existing within the UML specification.

In 2004, Bauerdick et al. investigated OCL WFRs specified in the UML 2.0 superstructure [BGG04]. They detected 246 OCL rules containing 361 errors. The errors were structured into five categories, namely (1) syntactical errors, (2) minor inconsistencies, (3) type checking errors, (4) general problems, and (5) inconsistencies with the UML-based Specification Environment (USE) [GBR07], the tool they used for OCL parsing and static semantics checking. The work was based on an earlier analysis done by the same group in 2000 [RG00] that analysed an excerpt of the UML 1.3 containing 71 OCL expressions with 39 errors. Although Bauerdick et al. investigated many syntactic and semantic errors, they did not conclude how to improve the UML specification process nor did investigate whether these errors could be avoided due to enhancements of the OCL.

In 2003, Fuentes et al. [FQL+03] investigated the consistency of OCL rules within the UML 1.5 specification. They tried to implement a UML case tool that supported a model checker based on the WFRs specified in the UML specification. When translating the WFRs manually into .Net code they identified about 450 errors they categorised into three categories: (1) non-accessible elements, (2) empty names and (3) other errors. Besides the identification of 450 errors Fuentes et al. also investigated inconsistencies (and even contradictions) between the given OCL rules and their textual documentation.

Chiorean et al. [CCP+02] focused in their investigations on conceptual errors in WFRs of the UML 1.3 metamodel in 2002. Based on their experiments in UML model checking with concrete UML tools they argue that syntactic and semantic errors are not sufficient to ensure a correct UML metamodel. There are also some conceptual—i.e., design—errors in the UML specification.

3 Methodology

For our analysis we used the UML 2.3 superstructure specification as published by the OMG [Obj10]. Unfortunately, between our study and the publication of its results, UML 2.4 reached its beta state and can now be considered as the latest official specification [Obj11d]. However, a short investigation of the constraints specified in the UML 2.4 superstructure specification showed that most of the investigated problems still exist for UML 2.4.

To parse and statically check the WFRs specified within the UML superstructure, we used the OCL parser/editor of Dresden OCL.[1] As the UML metamodel we used the Eclipse Modeling Framework (EMF)-based implementation of the Eclipse Model Development Tools (MDT).[2] The metamodel had minor differences to the specified UML 2.3 metamodel but our evaluation was not influenced by these differences.[3]

In our work, the OCL constraints were taken from the UML 2.3 superstructure specification as available from the OMG website [Obj10]. We investigated the OCL expressions marked as constraints only. Since only the OCL expressions were given in the specification, the context declarations had to be added manually. Constraints that were defined in a textual (English) form only were counted as defined constraints but not translated into nor parsed as OCL rules.[4] All constraints containing errors[5] have been investigated and where possible, the errors have been categorised and fixed. The same errors occuring multiple times within the same OCL rule were counted as one error. Different errors occuring within the same OCL rule were counted separately. Further OCL expressions used within the UML specification to define additional query operations where not considered during this study. However, spot checks confirmed that the results for the constraints presented in this paper can be assumed to being similar for the query operation. The EMF/Ecore-based UML metamodel together with all the WFRs as an OCL text file and an Excel spreadsheet containing the evaluation results can be obtained from the Dresden OCL website.[6]

4 Results

In the following we illustrate the results of our investigation. First, we present some general statistics of the WFRs defined in the UML 2.3 Superstructure in Subsection 4.1 and shortly discuss the complexity of these WFRs in Subsection 4.2. Afterwards in Subsection 4.3, we group the identified errors into five categories and give some examples

[1] http://www.dresden-ocl.org/

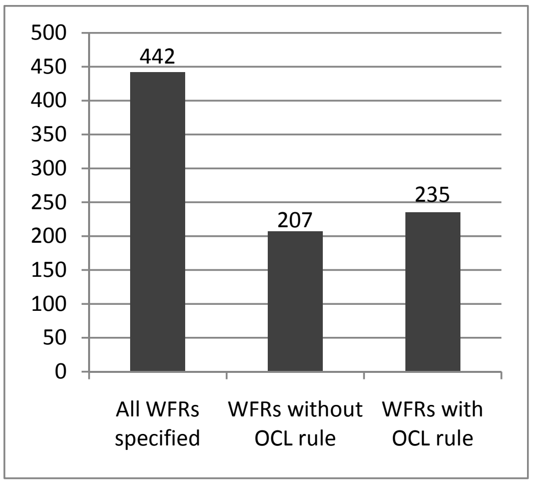
[2] http://www.eclipse.org/mdt/

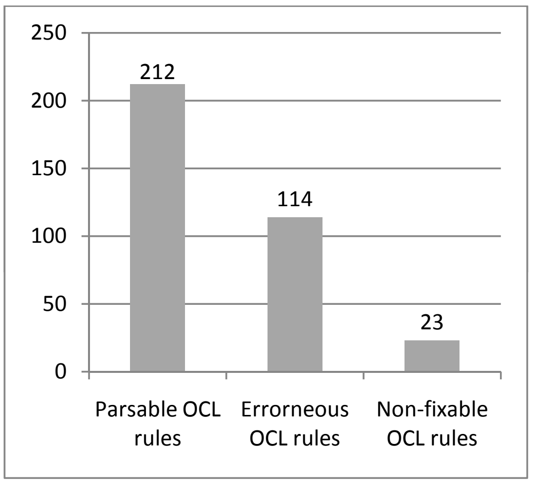
[3] The major difference to the formally released UML metamodel is that the MDT metamodel does not contain separate packages for the different parts of the UML, as for example the `PrimitiveTypes` and `Profiles` packages. Further minor differences result from detected issues implementing OMG's UML metamodel and have been submitted to the UML 2.x Superstructure and Infrastructure Revision Task Forces (RTFs), see http://www.omg.org/issues/uml2-rtf.open.html.

[4] It is not really clear why the UML specification defines almost half of all constraints only in English language. In [Obj10] it is written: *The Constraints sub clause contains a numerical list of all the constraints that define additional well-formedness rules that apply to this concept. Each constraint consists of a textual description and may be followed by a formal constraint expressed in OCL. Note that in a few cases, it may not be possible to express the constraint in OCL, in which case the formal expression is omitted.*

[5] According to the Dresden OCL parser that supports OCL 2.3.

[6] http://www.dresden-ocl.org/index.php/DresdenOCL:WFRsInUML





All WFRs specified (including non-OCL)	442	
WFRs without OCL rule	207	46.8% (of all WFRs)
WFRs with OCL rule (including non-parsable)	235	53.2% (of all WFRs)
Parsable OCL rules	212	90.2% (of all OCL rules)
OCL rules containing at least one error	114	48.5% (of all OCL rules)
Non-fixable errorneous OCL rules	23	9.8% (of all OCL rules)
Total number of found OCL errors	320	

Figure 1: General WFR statistics.

for these different types of errors. Finally, we shortly evaluate the usage of implicit conversions within UML 2.3's WFRs in Subsection 4.4.

4.1 General WFR Statistics

Figure 1 shows some general statistics for the WFRs defined in the UML 2.3 Superstructure. In total, the specification contains 442 WFRs. For 207 of these WFRs, no OCL constraints were specified. For the remaining 235 constraints, OCL rules exist. However, we were able to parse only 212 constraints, 23 OCL rules contained errors we were not able to fix. Of the specified OCL constraints 114 constraints were only parsable after modifications; i.e., they contained at least one error. These are 48.5% (!) of all specified OCL constraints. In total we found 320 errors within these 114 erroneous OCL rules.

4.2 WFR Complexity

Besides the number of constraints and errors we also tried to investigate the complexity of the specified OCL rules. With complexity we do not mean their complexity w.r.t. execution time but the complexity of their abstract syntax structure. Thus, we measured the number of expressions used in each constraint and also the depth of the constraint's

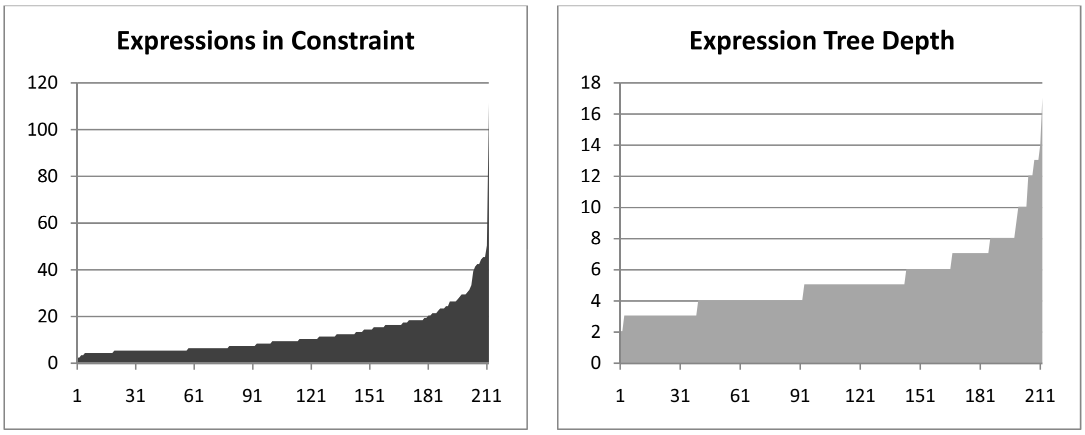


Figure 2: OCL WFR complexity: number of expressions in each constraint (left) and depth of expression tree in each constraint (right). Results are ordered by the metrics' values.

expression tree.[7] The results of our complexity analysis are shown in Figure 2. All 212 parsable OCL rules contained in total 2623 OCL expressions, which was an average of 12.37 expressions per constraint (min = 2, max = 111, median = 9). The tree depth of the rule's expressions ranged from 2 to 17; the average tree depth was 5.29 (median = 5). These results show that the most specified WFRs were rather simple. E.g, an expression count of two means either a simple property call (`self.property`) or a simple operation call (`self.operation()`). A few complex constraints exist (111 expressions, tree depth 17) such as the constraint shown in Listing 1 but both the average expression count and the median as well as the average and median tree depth were rather low. Although the shown constraint is not that complex w.r.t. its dynamic semantics, the example demonstrates that the UML 2.3 superstructure contains OCL expressions that should be refactored to improve their readability and comprehensibility.

4.3 Types of Investigated Errors

Figure 3 shows all errors found within the OCL WFRs of the UML 2.3 superstructure. As can be seen, we found 320 errors that we structured into 14 specific types of errors that we again grouped into five categories similar to the categories defined by Bauerdick et al. [BGG04]. Below, all found categories are shortly presented and examples for the found error types are given.

[7] The count and depth of the expressions was measured after OCL parsing and static semantics analysis which results in an abstract syntax model containing OCL expressions and their references to the constrained model [BD07].

```
/* From UML 2.3 superstructure, clause 17.2.1. */
(self.informationSource->forAll(p | p->oclIsKindOf(Actor)
  or oclIsKindOf(Node) or oclIsKindOf(UseCase)
  or oclIsKindOf(Artifact) or oclIsKindOf(Class)
  or oclIsKindOf(Component) or oclIsKindOf(Port)
  or oclIsKindOf(Property) or oclIsKindOf(Interface)
  or oclIsKindOf(Package) or oclIsKindOf(ActivityNode)
  or oclIsKindOf(ActivityPartition)
  or oclIsKindOf(InstanceSpecification)))
and
(self.informationTarget->forAll(p | p->oclIsKindOf(Actor)
  or oclIsKindOf(Node) or oclIsKindOf(UseCase)
  or oclIsKindOf(Artifact) or oclIsKindOf(Class)
  or oclIsKindOf(Component) or oclIsKindOf(Port)
  or oclIsKindOf(Property) or oclIsKindOf(Interface)
  or oclIsKindOf(Package) or oclIsKindOf(ActivityNode)
  or oclIsKindOf(ActivityPartition)
  or oclIsKindOf(InstanceSpecification)))
```

Listing 1: The WFR with maximum tree depth (17) and expression count (111).

4.3.1 Syntactical Errors

The first category contains syntactical errors. We identified six different types of syntactical errors which are (cf. Listing 2): (1) typing errors, (2) missing leading/closing brackets, (3) wrong or incomplete if statements, (4) missing escape characters for properties named like OCL keywords, (5) wrong uses of the # character in front of literals, and (6) wrong uses of the different navigation operators `.` and `->`. In total we analysed 78 syntactical errors in 24.3% of the specified WFRs.[8]

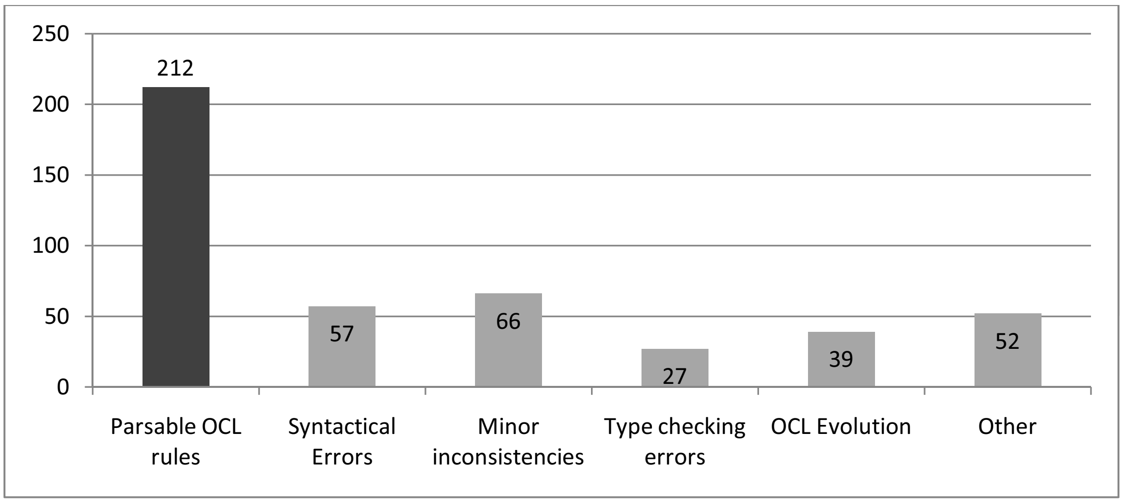
4.3.2 Minor Inconsistencies

The second category of errors are minor inconsistencies (cf. Listing 3): (1) references to `NamedElements` within the UML metamodel that did not exist or that had been replaced during the metamodel's evolution and (2) rules that called properties instead of operations (or vice versa). We investigated 71 errors within this category which were in 28.1% of all specified OCL rules. Thus, every fourth constraint contained such an error.

4.3.3 Type Checking Errors

A third category contains errors related to type nonconformance (cf. Listing 4): (1) invariants, not resulting in a `Boolean` value, (2) usage of wrong iterators, e.g., resulting

[8] The percentage differs from the total amount of 78 errors since some constraints contained multiple syntactical errors of different types.



Type of Error	**Occurrence**	**Infected Rules**
Syntactical Errors	**78**	**24.3%**
Typing errors	15	6.4%
Missing opening/closing brackets	27	11.5%
Wrong/incomplete if statement	8	3.4%
Missing escape character	14	6.0%
Wrong use of # in front of literal	6	2.6%
Wrong use of . or -> operator	9	3.8%
Minor Inconsistencies	**71**	**28.1%**
Wrong NamedElement referred	62	26.4%
Operation instead of property (or vice versa)	9	3.8%
Type Checking Errors	**48**	**11.5%**
Wrong result type (not boolean)	20	8.5%
Usage of wrong iterator	4	1.7%
Missing asSet()	15	6.4%
Missing asOrderedSet()	9	3.8%
OCL Evolution	**71**	**16.6%**
Correction of enumeration literals	54	23.0%
Use of Set{} instead of null	16	6.8%
Other	**52**	**22.1%**
Sum	**320**	–

Figure 3: OCL error classification.

```
/* From UML 2.3 superstructure, clause 15.3.6.
   (1) Spelling error: 'Psuedostate'
   (2) The last closing bracket is wrong. */
region->forAll (r | r.subvertex
  ->forAll (v | v.oclIsKindOf(Psuedostate) implies
    ((v.kind <> #deepHistory) and
      (v.kind <> #shallowHistory)))))

/* From UML 2.3 superstructure, clause 7.3.3.
   (3) Implication is meant here.
       The if-statement requires both else and endif. */
  if memberEnd->size() > 2
  then ownedEnd->includesAll(memberEnd)

/* From UML 2.3 superstructure, clause 11.3.36.
   (4) context is a keyword -> use _context. */
self.context->size() = 1

/* From UML 2.3 superstructure, clause 11.3.14.
   (5) #false instead of false. */
self.association().isAbstract = #false

/* From UML 2.3 superstructure, clause 15.3.11.
   (6) isEmpty() must be called using -> */
isSimple = region.isEmpty()
```

Listing 2: Examples for syntactical errors.

```
/* From UML 2.3 superstructure, clause 11.3.11.
   (1) object.multiplicity.is(1,1) evolved to object.is(1,1). */
self.object.multiplicity.is(1,1)

/* From UML 2.3 superstructure, clause 15.3.8.
   (2) PropertyCall instead of OperationCall: size. */
(self.kind = #initial) implies
(self.outgoing->size <= 1)
```

Listing 3: Examples for minor inconsistencies.

in a `Bag` of `Booleans` instead of a single `Boolean` value, missing `asSet()` invocations[9] (3) and missing `asOrderedSet()` invocations (4). In total we investigated 48 type checking errors contained in 11.5% of all specified OCL rules.

4.3.4 OCL Evolution

A fourth category contains errors that are related to the evolution of the OCL. This category contains two types of errors (cf. Listing 5): (1) enumeration literals that are referred using a # character instead of their enumeration type (which was allowed in OCL 1.3 (cf. [Obj00, clause 7.4.2]) but is not allowed in OCL 2.x anymore (cf. [Obj11b, clause 7.4.2])) and (2) the usage of empty collections instead of returning a `null` value (`Undefined-` or `NullLiterals` are supported since OCL version 2.2). In total we found 71 errors of this category within 16.1% of all specified OCL rules.

4.3.5 Other Errors

The fifth and last category contained all errors that were too specific to categorise them as own types of errors (cf. Listing 6). 52 constraints (22.1% of all OCL rules) contained such errors.

4.3.6 Non-fixable Errors and missing OCL Rules

For completeness reasons we also give examples for constraints we were not able to fix w.r.t. their static semantics (cf. Listing 7, (1)) and WFRs for which the UML superstructure does not define any OCL expression at all (cf. Listing 7, (2) and (3)).

4.4 Implicit Conversions

Besides the classification of errors we also investigated how many OCL rules use implicit conversions as (1) the implicit `asSet()` conversion and (2) the implicit `collect()` iterator (cf. Listing 8). Within the 212 parsable OCL rules we identified 94 usages of the implicit `asSet()` conversion and 18 usages of the implicit `collect()` iterator. We analysed implicit conversions because they often cause problems explained below in Section 5.2.

5 Lessons Learnt

After presenting our analysis results we evaluate the resulting statistics. We also discuss possible improvements of the UML specification process and OCL's concrete syntax and standard libary.

One of the biggest surprises for us was that our results were very similar to the results presented by Bauerdick et. al in 2004 [BGG04]. Although some OCL constraints have been added to newer versions of the UML 2.x specification, these similarities show that

[9] Although the `asSet()` operation is defined as being implicitely in OCL, this does not work in all situations and hence, sometimes `asSet()` has to be called explicitely.

```
/* From UML 2.3 superstructure, clause 11.3.1.
   (1) Results in Bag(Boolean) since event is a collection
       (implicit collect()). */
trigger.event.oclIsKindOf(CallEvent)

/* From UML 2.3 superstructure, clause 7.3.4.
   (2) forAll must be used instead of collect. */
self.endType->excludes(self) and self.endType
  ->collect(et|et.allparents()->excludes(self))

/* From UML 2.3 superstructure, clause 7.3.4.
   (3) Implicit asSet() on ownedEnd does not work. intersection()
       expects a Set as its argument. */
ownedAttribute->intersection(ownedEnd)->isEmpty()
```

Listing 4: Examples for type checking errors.

```
/* From UML 2.3 superstructure, clause 7.3.15.
   (1) VisibilityKind::public/private must be used. */
self.visibility = #public or self.visibility = #private

/* From UML 2.3 superstructure, clause 7.3.36
   (2) Null must be used instead of Set{}. */
lower = if returnResult()->notEmpty() then returnResult()
  ->any().lower else Set{} endif
```

Listing 5: Examples for OCL evolution errors.

```
/* From UML 2.3 superstructure, clause 7.3.33.
   .concat() must be used instead of ->union() (2x). */
(self.name->notEmpty() and self.allNamespaces()
  ->select(ns | ns.name->isEmpty())->isEmpty()) implies
    self.qualifiedName = self.allNamespaces()
      ->iterate( ns : Namespace; result: String =
        self.name | ns.name->union(self.separator())
          ->union(result))
```

Listing 6: Example for other OCL errors.

```
/* From UML 2.3 superstructure, clause 11.3.21.
   (1) The property connection is not defined for LinkActions. */
self.endData->collect(end) = self.association()->collect(connection)

/* From UML 2.3 superstructure, clause 7.3.10.
   (2) No OCL expression specified (not possible). */
/* Evaluating the value specification for a constraint must not have
   side effects. */
-- Cannot be expressed in OCL.

/* From UML 2.3 superstructure, clause 9.3.4.
   (3) No OCL expression specified (ommitted). */
/* All the client elements of a roleBinding are in one classifier and
   all supplier elements of a roleBinding are in one collaboration and
   they are compatible. */
-- No OCL specified.
```

Listing 7: Examples for non-fixable OCL errors and missing OCL rules.

```
/* From UML 2.3 superstructure, clause 15.3.7.
   (1) Implicit asSet() on effect. */
effect->isEmpty()

/* From UML 2.3 superstructure, clause 11.3.23
   (2) Implicit collect() on qualifier. */
self.value->excludesAll(self.qualifier.value)
```

Listing 8: Examples for implicit conversions.

many WFRs in the UML specification have not been revised or improved within the last seven years!

All five categories of errors presented above contain many errors (cf. Table 3). Every fourth constraint contains syntactical errors and minor inconsistencies. Every tenth constraint is inconsistent w.r.t. to its result type, every sixth constraint contains inconsistencies caused by OCL's syntax evolution.

5.1 Necessary Improvements of the UML Specification Process

The presented statistics show that the UML specification process as performed until today is definitely insufficient. It is not very helpful to specify constraints as WFRs if neither syntactical nor static semantics checks on them are performed. Although the constraints can give impressions about the WFRs of UML, in many cases their meaning is unclear or imprecise (e.g., we were not able to fix 23 of the erroneous OCL constraints although we used an OCL parser for assistence). The evolution of the UML metamodel seems to be another major problem which shows that the WFRs must coevolve every time the metamodel is modified or extended. These results lead to the fact that the UML specification process must be altered. In this Section we propose four major steps how the specification process could be altered. These are: (1) change to model-based specifications, (2) use of elucidative programming technique, (3) use of UML/OCL coevolution, and (4) checks of the WFR's semantics using OCL unit testing. All of them are shortly presented below.

5.1.1 Model-Based UML Specification.

The first and most obvious change is that the UML specification process should be model-based. As model-based we consider a specification process that is based on a well-defined UML metamodel and well-defined, syntactically, and semantically checked OCL WFRs. Although the UML metamodel is already defined and diagrams from modelling tools are imported into the specification, this process obviously does not include the OCL WFRs. If the OCL rules were parsed and checked for correctness using existing OCL tools, inconsistencies with changes performed in the UML metamodel could be detected and removed before each revised UML specification is formally released.

5.1.2 Elucidative Specification Process.

Since the UML specification relates to many models, diagrams and constraints that are obviously model-based, we propose to use techniques from *elucidative programming* [Nor02] for the UML specification process. Elucidative programming proposes to interconnect and semi-automate the process of programming and code documentation. Documentations are based on documents linking to code snippets that can be semi-automatically updated if the related code was modified. If updates are not possible automatically, warnings in the documentation document can help developers to synchronise documentation and code. We think that similar techniques can also be applied to documents like specifications that do not include code but many references to models, diagrams and constraints (that can be considered as code). The application of such a process for the UML specification would allow automated updates of the specification since all model-based content would be only hooked into and referred from the specification document. Modifications of the models would be propagated automatically to the specification and where updates would not be possible automatically, hints like warnings could help to propagate the changes manually. One example for a tool supporting the required automation for *elucidative specification* is the Development Environment For Tutorials (DEFT) [Bar09] tool.[10] Although originally developed for code documentation and the creation of software framework tutorials, DEFT can also be used to perform elucidative techniques on EMF-based models. Since EMF-based versions of the UML metamodel and EMF-compatible OCL tools exist, DEFT could be used for an elucidative UML specification process.

[10] http://deftproject.org/

5.1.3 Support for Coevolution of UML and OCL.

Once a model-based representation of both a UML metamodel and the OCL-based WFRs would be used within the UML specification process, another improvement of the process would be the usage of coevolution tooling for UML and OCL. OCL-coevolution (or co-refactoring) tries to synchronise the OCL rules with the constrained metamodel whenever a change in the metamodel is performed. Of course, the easiest way would be a simple reparsing of all OCL constraints after every metamodel modification. However, automate or semiautomate coevolution would improve this process w.r.t. its time effort. Where coevolution is not possible automatically, guidance could help to synchronise the WFRs and the metamodel. First theoretical works exist in the domain of UML-OCL coevolution [MB05, HSLF11] and seem to be promising for future UML specifications.

5.1.4 OCL Unit Tests for the WFR's Dynamic Semantics.

Besides parsing the OCL constraints and checking their static semantics, their dynamic semantics is another challenge of the current UML specification.[11] A constraint that has correct static semantics can still be erroneous, as some aspects like termination of recursive operations, absence of side effects (e.g., invocation of a model-defined operation having side effects) and the avoidance of invalid results (e.g., division by zero) cannot be ensured statically. Furthermore, its difficult to check statically whether a constraint expresses what it shall express. Thus, we propose to define OCL unit tests for the WFRs of the UML specification. These unit tests should consist of both positive and negative test cases for all defined WFRs using instances of the UML metamodel (i.e., UML models). Once defined, these tests could be shipped together with the UML specification and even used as a regression test suite for UML tools under development. Although we are aware of the fact that tests cannot totaly avoid errors within programs (and thus within executable OCL expressions as well), we argeue that testing OCL expressions using unit tests might improve their quality and also might help to avoid many semantical errors within these expressions. First works in the domain of OCL unit testing have been done by Chimiak-Opoka et al. [CO09] and by Hamann et al. [HG10] and could build a basis for UML WFR unit tests.

5.2 Sensible Improvements of the OCL

A further lesson learnt is the possible improvement of OCL itself. Therefore we propose some OCL improvements that could be done to avoid similar specification errors within future versions of the UML WFRs and the WFRs of other languages and metamodels. Since we investigated only WFRs specified on one metamodel it is hard to generalise the assumptions we present here. However, we think that the proposed improvements are general enough to be appropriate for other metamodels and use cases of the OCL as well. Neverless, we do not claim that the following enumeration is complete nor

[11] As WFRs are specified at the metamodel level, they are validated during modelling of UML models. Thus by the using term *dynamic semantics* we are referring to the constraint's evaluation (and thus, their execution) during modelling by using a UML case tool.

```
1 /* Two ways to invoke size() on Strings. */
2 name.size() > 1
3 name->size() > 1 -- means Set{name}->size()!
4
5 /* Two ways to invoke asSet() on objects. */
6 name.asSet()
7 name->asSet() -- means Set{name}->asSet()!
```

Listing 9: Example for semantic -> confusion.

sensible for all use cases of OCL. Furthermore, we do not claim that all WFRs defined within the UML 2.3 superstructure can be expressed with the proposed modifications and extensions. E.g., the constraint (2) shown is Listing 7 cannot not be expressed in OCL as it forbids side effects for OCL's dynamic semantics.[12]

5.2.1 Remove the -> Navigation Operator.

One of the most confusing concepts in OCL is the separation of operation calls on collections and single elements in its concrete syntax. For operation calls on collections the `->` notation is used, whereas for operation calls on single elements the `.` notation must be used. Our investigation showed that even the designers of the UML—which should be very familiar of the OCL and its usage—did not fully understand this concept since nine constraints contained errorneous usages of these navigation notations. Furthermore, we think that the separation of these two notations can lead to major semantic inconsistencies. Imagine a modeller that wants to express that a property `name` within one of his/her constraints should contain at least two characters. He/she might define the rule as shown in Listing 9, line 2. However, a short modification of this rule as shown in line 3 of the same listing, has a total different meaning. Now, the `name` is implicitly converted into a `Set` containing one `String` (if `name` is not `null`) and the `size()` operation will never result in a value greater than one! Another example is shown in the lines 6–7 of Listing 9. In both lines the `asSet()` operation is explicitly invoked on `name`. However, in line 7, the wrong navigation operator is used and hence the statement is parsed to `Setname->asSet()` which causes an unecessary conversion and thus, overhead during execution. These simple examples show, how much impact a simple notation confusion can have on a constraint's dynamic semantics. Thus, we propose *not* to use the `->` notation for collection-based operation calls and modify the syntax to the `.` notation instead. However, iterators are expressed using the `->` notation as well. For iterators it might be appropriate to preserve the `->` notation to explicitly separate iterator from operations calls as iterators are a separate concept within the OCL.

[12] Whether or not such a constraint is sensible at all as the OCL specification forbids side effects for OCL expressions is out of scope of this discussion.

```
 1 /* From UML 2.3 superstructure, clause 17.2.1.
 2    Implicit, unnecessary asSet conversion of p and unnecessary
 3    collect on implicit asSet in expression p->oclIsKindOf(Class).*/
 4 self.conveyed.represented->forAll(p | p->oclIsKindOf(Class)
 5   or oclIsKindOf(Interface) or oclIsKindOf(InformationItem)
 6   or oclIsKindOf(Signal) or oclIsKindOf(Component))
 7
 8 /* From UML 2.3 superstructure, clause 18.3.6.
 9    Implicit collect on metamodelReference, importedPackage,
10    elementImport, importedElement (2x), metaclassReference. */
11 self.metamodelReference.importedPackage.elementImport
12   .importedElement.allOwningPackages()
13     ->union(self.metaclassReference.importedElement
14       .allOwningPackages())->notEmpty()
```

Listing 10: Examples for troubles using implicit asSet() and collect().

5.2.2 Avoid implicit asSet() and implicit collect().

Related to the problem of the `->` notation discussed above, we think that the implicit usage of `asSet()` and `collect()` should be avoided, or even forbidden. The examples of Listing 9 presented above were only possible because the `name` was implicitly converted into a `Set` in the lines 3 and 7. The investigation of the WFRs of UML 2.3 showed that modellers are often unfamiliar were implicit `asSet()` conversions occur and do not consider them when specifying OCL constraints. The same phenomenon can be observed for the implicit `collect()` iterator (cf. Listing 10). The example of Listing 10, lines 8–14 demonstrates that modellers are often not aware where they use the implicit `collect()` iterator. Although probably sensible in some situations the example shows that an unknown usage of `collect()` can cause many chained iterations on collections that can cause large performance bottlenecks during the constraint's evaluation. Thus, we think that the usage of implicit `collect()` should be avoided—at least within the WFRs of an official specification such as the UML to improve clarity and readability of the given WFRs.

5.2.3 Introduce a selectByKind() Iterator.

Another pattern that often occurred within the specified WFRs (and which is typical for WFRs in general) was an iteration on the elements of a collection filtering all elements having a specific type, followed by a downcast of these elements to the filtered type (cf. Listing 11). We found 19 WFRs containing this pattern; often the downcast was missing which led to some of the reported type errors. Thus, we think the introduction of an iterator that does both type check and downcast within the same iteration would be useful (cf. Listing 12).[13] Such an iterator would simplify the specification of many WFRs for

[13] The specific semantics for all cases is not specified here. E.g., the result of the `selectByKind()` iterator for elements not matching the Type. Do they lead to an invalid result or an empty collection? Both

```
/* From UML 2.3 superstructure, clause 7.3.3. */
parents()->select(oclIsKindOf(Association))
  .oclAsType(Association)->forAll(p |
    p.memberEnd->size() = self.memberEnd->size())
```

Listing 11: Example downcast after type-related filtering of elements.

```
/* Modified from UML 2.3 superstructure, clause 7.3.3. */
parents()->selectByKind(Association)->forAll(p |
  p.memberEnd->size() = self.memberEnd->size())
```

Listing 12: Example for the proposed selectByKind() iterator.

many metamodels since navigation, filtering and downcasting is a common problem during navigation of elements within a model. Of course in analogy to the standard library operations `oclIsKindOf()` and `oclIsTypeOf()` a second iterator `selectByType()` could be introduced as well to differentiate between hard and soft type checks during filtering. A similar construct called `collectselect()` allowing to filter and modify elements of a collection within the same iteration has already been introduced into Query/View/-Transformation Operational (QVTOperational) which extends the OCL for its usage as a transformation language [Obj11a].

6 Conclusion

In this paper we presented an analysis of the OCL WFRs specified in the UML 2.3 Superstructure. All specified constraints have been parsed and checked for static semantics correctness based on standard compliant UML and OCL implementations.[14] As UML 2.4 reached its beta status during our study [Obj11d], we exemplarily checked that all examples presented in this paper have not been revised within the newest version of UML and thus, still exist. We published all OCL WFRs of the UML 2.3 Superstructure including the modified OCL expressions for the effortless reuse in the further OMG UML 2.x specification process. We have shown that many of the specified constraints contain syntactical and type-related errors (48.5%) as well as inconsistencies to the UML metamodel (26.4%) or the latest OCL syntax (16.6%). Thus, we discussed proposals how to improve the UML specification process for future versions w.r.t. to the WFR problems investigated in this paper. We proposed a model-based specification process that exploits the tooling existing today for both UML modelling and OCL parsing. Further we proposed the use of elucidative techniques as well as coevolution tooling for UML and OCL. Although these proposals were all related to the UML specification process we think that

semantics would be appropriate and helpful in some situations.

[14] Some minor inconsistencies exist in both the EMF-based UML metamodel and Dresden OCL's OCL implementation. However, they did not affect the results of our analysis.

similar steps could be applied to other language specification processes as well. Finally we gave hints how the OCL's concrete syntax and the OCL standard library could be further improved w.r.t to WFR specifications.

For future work we plan to investigate WFRs specified on other metamodels as well as to evaluate the applicability of the proposed changes of a language's specification process. Furthermore it would be interesting which of the constraints providing no or only wrong OCL expressions have been implemented by the static semantics of existing UML case tools such as Eclipse MDT and how these rules could be extracted from them for future versions of the UML 2.x specification.

Acknowledgements

This research has been co-funded by the European Social Fund and Federal State of Saxony within the project ZESSY #080951806. We like to thank all persons that have been involved into the development of Dresden OCL. Especially we like to thank Joanna Chimiak-Opoka, Michael Thiele and Lars Schütze for their helpful feedback during discussing the topics of this paper. Further interesting and helpful comments arose from the reviewers' feedback during the paper's submission and reviewing process.

Bibliography

[Bar09] A. Bartho. Creating and maintaining tutorials with deft. In *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. Pp. 309–310. 2009.

[BD07] M. Bräuer, B. Demuth. Model-Level Integration of the OCL Standard Library Using a Pivot Model with Generics Support. In *Ocl4All: Modelling Systems with OCL Workshop at MoDELS 2007*. Technische Universität Berlin, Berlin, Germany, October 2007.

[BGG04] H. Bauerdick, M. Gogolla, F. Gutsche. Detecting OCL Traps in the UML 2.0 Superstructure: An Experience Report. In Baar et al. (eds.), *UML 2004 - The Unified Modelling Language*. Lecture Notes in Computer Science 3273, pp. 188–196. Springer Berlin / Heidelberg, 2004.

[CCP$^+$02] D. Chiorean, A. Carcu, M. Pasca, C. Botiza, H. Chiorean, S. Moldovan. UML Model Checking. *Informatica* XLVII:71–88, 2002.

[CO09] J. Chimiak-Opoka. OCLLib, OCLUnit, OCLDoc: Pragmatic Extensions for the Object Constraint Language. In Schürr and Selic (eds.), *Model Driven Engineering Languages and Systems*. Lecture Notes in Computer Science 5795, pp. 665–669. Springer, Berlin / Heidelberg, 2009.

[FQL$^+$03] J. Fuentes, V. Quintana, J. Llorens, G. Génova, R. Prieto-Díaz. Errors in the UML metamodel? *ACM SIGSOFT Software Engineering Notes* 28(6), 2003.

[GBR07] M. Gogolla, F. Büttner, M. Richters. USE: A UML-based specification environment for validating UML and OCL. *Science of Computer Programming* 69(1-3):27–34, 2007.

[HDF02] H. Hussmann, B. Demuth, F. Finger. Modular architecture for a toolset supporting OCL. *Sci. Comput. Program.* 44:51–69, July 2002.

[HG10] L. Hamann, M. Gogolla. Improving Model Quality by Validating Constraints with Model Unit Tests. In *Proceedings of the Models Workshop on Model-Driven Enginering, Verification and Validation (MoDeVVa2010)*. 2010.

[HSLF11] K. Hassam, S. Sadou, V. Le Gloahec, R. Fleurquin. Assistance System for OCL Constraints Adaptation During Metamodel Evolution. In Mens et al. (eds.), *Proceedings of 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*. Pp. 151–160. Conference Publishing Services (CPS), Los Alamitos, California, USA, 2011.

[MB05] S. Marković, T. Baar. Refactoring OCL Annotated UML Class Diagrams. In Briand and Williams (eds.), *Model Driven Engineering Languages and Systems*. Lecture Notes in Computer Science 3713, pp. 280–294. Springer Berlin / Heidelberg, 2005.

[MDT11] Eclipse Model Development Tools: OCL. 2011.
http://www.eclipse.org/modeling/mdt/?project=ocl

[Nor02] K. Normark. Requirements for an elucidative programming environment. In *Proceedings of the 8th International Workshop on Program Comprehension (IWPC 2000)*. Pp. 119–128. 2002.

[Obj00] Object Management Group (OMG). Unified Modeling Language, Version 1.3. Online available specification, March 2000.
http://www.omg.org/spec/UML/1.3/

[Obj03] Object Management Group. Common Warehouse Metamodel (CWM) Specification. Version 1.1. March 2003.
http://www.omg.org/spec/CWM/

[Obj06] Object Management Group. Meta Object Facility (MOF) Core Specification, Version 2.0. January 2006.
http://www.omg.org/spec/MOF/2.0/

[Obj10] Object Management Group (OMG). Unified Modeling Language: Superstructure Version 2.3. May 2010.
http://www.omg.org/spec/UML/2.3/

[Obj11a] Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/-Transformation Specification (QVT), Version 1.1. January 2011.
http://www.omg.org/spec/QVT/

[Obj11b] Object Management Group. Object Constraint Language. Version 2.3, BETA2. Online available specification, March 2011.
http://www.omg.org/spec/OCL/2.3/Beta2/

[Obj11c] Object Management Group. Semantics of a Foundational Subset for Executable UML Models (fUML), Version 1.0. Online available specification, February 2011.
http://www.omg.org/spec/FUML/1.0/

[Obj11d] Object Management Group (OMG). Unified Modeling Language: Superstructure Version 2.4, BETA2. Online available specification, March 2011.
http://www.omg.org/spec/UML/2.4/

[Obj11e] Object Management Group (OMG). Unified Modeling Language (UML), OMG Formally Released Versions of UML. 2011. Visited in May 2011.
http://www.omg.org/spec/UML/

[RG00] M. Richters, M. Gogolla. Validating UML Models and OCL Constraints. In *Proceedings of the 3rd International Conference on The Unified Modeling Language: Advancing the Standard*. Pp. 265–277. Springer, Berlin Heidelberg, Germany, 2000.