

OCL FTF Issues on OCL 2.0 Revised Submission Version 1.6 January 6, 2003

Sender of a Message	2
ocllsNew for a collection	2
status of objects and tuples	2
Omit predefined type OclModelElement.	3
Consider OclType as a powertype.	3
Template return types in operation signatures.	3
Up- and Down-casts with oclAsType().	3
Lack of operation specifications.	3
Additional annotations in the OCL Standard Library.	4
domain for library operations /, div	4
Exception of strict evaluation (implies)	4
Exception of strict evaluation (forAll, exists)	4
Exception of strict evaluation (queries)	5
Exception of strict evaluation (=)	5
Flagging insecure cast from Set to Sequence	6
Flagging recursive definitions	6
Operator precedence	6
Attributes and Association Ends versus Properties	6
Undefined values, isEmpty() and Collections	7

Sender of a Message

Author: Alfred Strohmeier (alfred.strohmeier@epfl.ch)

Description: Provide access to the sender of a message

Rationale:

Consider the operation:

```
Account::withdraw (amount: Money)
```

Suppose a Person object sends the operation, and we want to state that the person has to be the owner of the account. Access to the sender of the message is needed. One might for instance imagine that the concrete syntax defines a keyword sender, and we could then write:

```
context: Account::withdraw (amount: Money)
```

```
pre: sender = self.owner
```

oclIsNew for a collection

Author: Alfred Strohmeier (alfred.strohmeier@epfl.ch)

Description: Provide an operation for creating a collection of new objects

Rationale:

Consider the case where you want to create a new smartcard for a set of persons. Any solution is currently longwinded. It would be nice to be able to have an operation oclIsNew that applies to a collection (or perhaps only to a Set), and states the number of objects to be created, e.g.:

```
let: s: Set(SmartCards) in s.oclIsNew(self.person->size())...
```

status of objects and tuples

Author: Alfred Strohmeier (alfred.strohmeier@epfl.ch)

Description: Provide a notation for the status of an object

Rationale:

It would be convenient to have a notation for denoting the status of an object. The type of such a status is a tuple. With such a notation it would be possible to compare the status of two objects or to compare the status of an object with a tuple. If not available, comparisons have to be performed on an attribute by attribute basis. Consider e.g.

```
p, p1 and p2 are Person(s)
```

```
p1.all = p2.all -- the 2 persons have same status, i.e.
```

```
is nicer and less error-prone than comparing all attributes:
```

```
p1.firstName = p2.firstName and p1.name = p2.name and ...
```

It would also be possible to compare with a tuple:

```
p.all = Tuple = Tuple {firstName = 'Alfred', name = 'Strohmeier', ...}
```

Omit predefined type `OclModelElement`.

Author: Stephan Flake (flake@c-lab.de)

Description: `OclModelElement` is currently defined in the OCL Standard Library. It can simply be omitted, as it is actually never used. `OclModelElementType`, however, must remain in the metamodel, e.g., for the mapping of `OclState`.

Consider `OclType` as a powertype.

Author: Stephan Flake (flake@c-lab.de)

Description: `OclType` is currently a subtype of `OclAny` and seen as an enumeration type. This leads to some drawbacks w.r.t. the specification of operations like `oclAsType()`, `oclIsKindOf()`, and `oclIsTypeOf()`, that need to reason about type conformance. As this cannot be done without accessing the metalevel, `OclType` should rather be considered as a powertype (cf. OCL Workshop paper at UML 2003: S.Flake, `OclType - A Type or Metatype?`).

Template return types in operation signatures.

Author: Stephan Flake (flake@c-lab.de)

Description: At some places, template parameter `T` appears in operation signatures, e.g., `oclAsType(typename:OclType) : T` (e.g., Sect. 6.2.1). At other places, this is denoted by "instance of `OclType`" or <<the return type of the invoked operation>>. It would be more meaningful when these informal return type descriptions are replaced by "`OclAny`". An additional constraint about the actual return type should be given when necessary.

Up- and Down-casts with `oclAsType()`.

Author: Stephan Flake (flake@c-lab.de)

Description: This is not treated consistently throughout the document. As the formal semantics already allows both up- and downcasts, this should also be allowed in Sect. 2.4.6.

Lack of operation specifications.

Author: Stephan Flake (flake@c-lab.de)

Description: Some operation specifications are still missing (they are marked by --TBD), e.g., `oclAsType()`. For this operation, a proposed specification is as follows (provided that `OclType` is a powertype):

```
1: context OclAny::oclAsType(typename:OclType) : OclAny
2: post: if OclType.allInstances()
3:     ->select(t:OclType | self.oclIsTypeOf(t))
4:     ->exists(t:OclType | typename.conformsTo(t) or t.conformsTo(typename)) then
5:     result = self and result.oclIsTypeOf(typename)
6:     else
7:     result = OclUndefined and result.oclIsTypeOf(OclVoid)
8:     endif
```

For a comparison, a complex OCL specification for ENUMERATION TYPE `OclType` can be found in the paper "`OclType - A Type or Metatype?`".

Additional annotations in the OCL Standard Library.

Author: Stephan Flake (flake@c-lab.de)

Description: The OCL Standard Library type system should make use of the notation offered by the official UML specification. In particular, abstract types (like `OclAny`, `Collection(T)`), datatypes (`Integer`, `Set(T)`), and enumeration types (`OclState`) can be denoted in italics and stereotyped, respectively.

An ellipsis can be used to indicate that further types are imported from a referred UML user model.

Moreover, `OrderedSet(T)` is missing in the OCL Standard Library Type system.

domain for library operations `/`, `div`

Author: Thomas Baar (thomas.baar@epfl.ch)

Description: clarify, whether `x/0` is undefined

Rationale: On page 6/6, 6-7 the semantics of operation `/` is described informally as 'The value of self divided by r (respective i).' It remains unclear what `self / 0` evaluates to.

On page 6/7 the `div`-operation is specified in terms of `/`-operation, however with a pre-condition `pre: i <> 0`.

Why is `div` handled differently from `/` ?

Exception of strict evaluation (`implies`)

Author: Thomas Baar (thomas.baar@epfl.ch)

Description: Exception from strict evaluation for `IMPLIES` is incomplete and contradicts set-theoretical semantics

Rationale: On page 2-10 only one exception from strict evaluation for `IMPLIES` is given:

False `IMPLIES` `x == True`

However, based on the official semantics of `IMPLIES` given on page A-12

also `x IMPLIES True == True`

Exception of strict evaluation (`forall`, `exists`)

Author: Thomas Baar (thomas.baar@epfl.ch)

Description: Exception of strict evaluation should be extended to `forall`, `exists`

Rationale: Suppose `r(o1) = undef`, `r(o2) = false` What is value of `Exp = {o1, o2}->forall(x| r(x))` ? One could argue, because of strict evaluation, the value of `Exp` is `undef`. However, this would contradict the semantics of `forall` als 'iterated and' given on page A.28. Similarly, for `exists`.

A note should be added on page 2-10 on evaluation of expressions based on `iterate`.

Exception of strict evaluation (queries)

Author: Thomas Baar (thomas.baar@epfl.ch)

Description: Strict evaluation for queries yields to contradictions in specifications

Rationale: Queries can be specified in two ways, as invariants and in form of pre/post conditions. Suppose we specify query $q(\text{arg})$ as

```
post: if (arg.oclIsUndefined()) then result = true else result =
false endfi
```

Having this, the following invariant should evaluate always to true:

```
self.q(arg) = true or self.q(arg) = false.
```

However, the invariant evaluates to undef once arg evaluates to undef thanks to strict evaluation.

There is a misconception of strict evaluation when it comes to queries. The idea of queries is to have user-defined functions on classes. Why should the user be restricted only to such function which return undef once one of its arguments is undef? Using OCL, the user can even specify queries which can handle undefined arguments (e.g. see post specification of $q(\text{arg})$). Obviously, the post specification for $q(\text{arg})$ makes sense.

The rule of strict evaluations for queries should be weakened to the case where the owner of the query (the object upon the query was called) is undefined.

Exception of strict evaluation (=)

Author: Thomas Baar (thomas.baar@epfl.ch)

Description: contradiction for evaluation of navigation expression

Rationale: Suppose to have two classes A, B and an association with multiplicity

0..1 on B

between them.

The invariant context

```
A inv: self.b = self.b
```

is evaluated for an instance of A not having an associated instance of B to

i) true, when the expression self.b has the type Set(B), because self.b is evaluated to emptyset and emptyset = emptyset is evaluated to true

ii) undef, when the expression self.b has the type B, because self.b is evaluated to undef and undef = undef is evaluated to undef thanks to strict evaluation of '='

This is a contradiction since the expression self.b can be both of type set(B) and B!

The examples also shows, that $x = x$ is not a tautology unlike in almost all other logics including classical predicates logic. This is especially confusing because OCL claims to be based on classical predical logic!

Flagging insecure cast from Set to Sequence

Author: Jörn Guy Süß (jgsuess@cs.tu-berlin.de)

Description: Interpreter to warn of nondeterministic cast

Rationale:

If an OCL expression contains a cast from Set to Sequence types, nondeterminism is introduced through the order of the new sequence. Either such casts should be disallowed, or the specification should require that implementations give feedback that nondeterministic behavior is to be expected.

Flagging recursive definitions

Author: Jörn Guy Süß (jgsuess@cs.tu-berlin.de)

Description: Interpreter to warn of recursive definitions

Rationale:

While recursive definitions are necessary to express certain constructs, they may lack a fixpoint. Specification should require implementations to provide either an occurs check or structured time-out/stack-trace to handle these situations.

Operator precedence

Author: Octavian Patrascoiu (O.Patrascoiu@kent.ac.uk)

Description: Logical operators 'and', 'or', and 'xor' have the same precedence, which in my opinion is not natural. I think that the precedence of these operators should be from highest to lowest as follows:

'and'

'or'

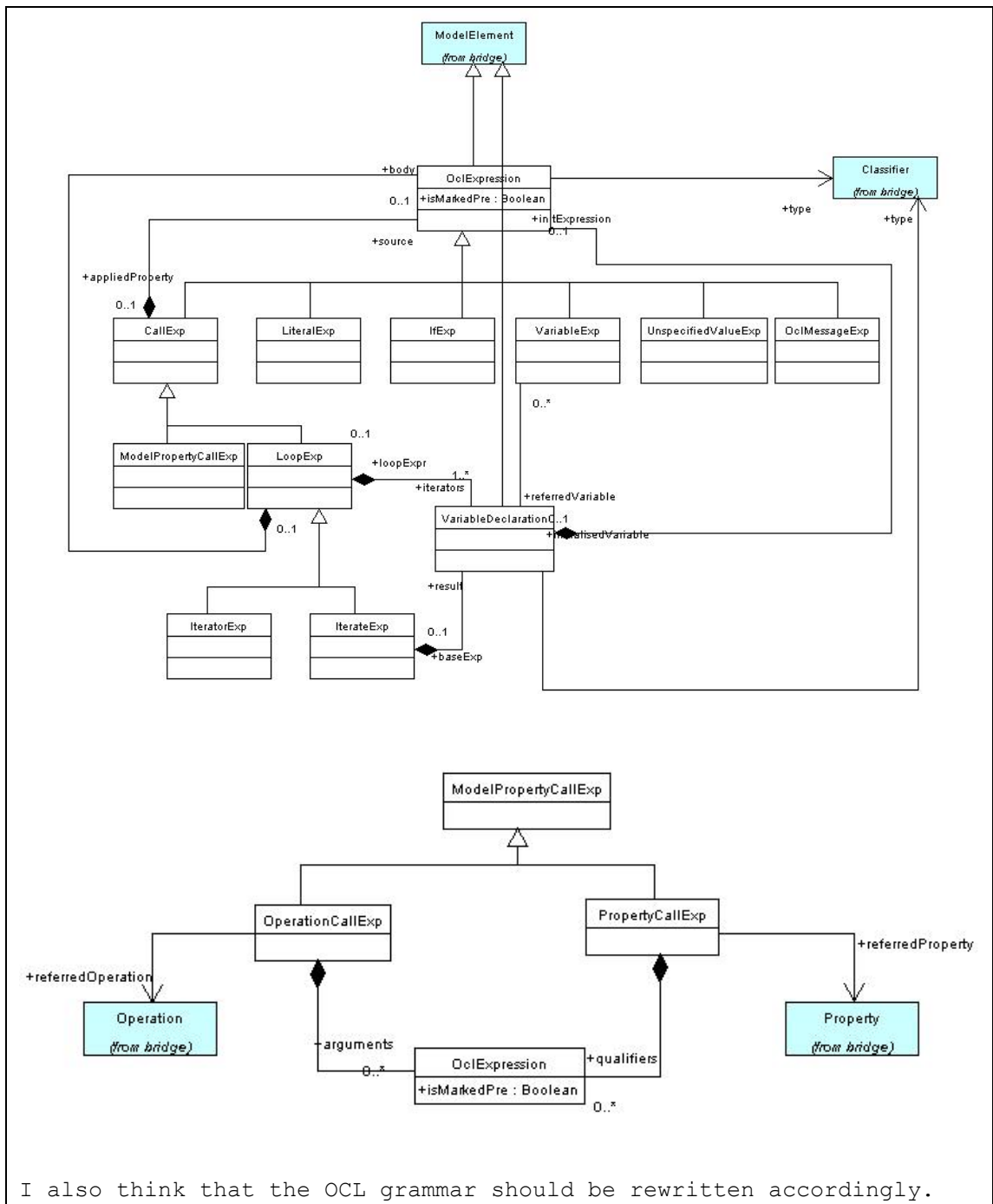
'xor'

Also, the precedence of some useful operators like 'div', 'mod', '^', and '^', is not specified in section 4.3.2.

Attributes and Association Ends versus Properties

Author: Octavian Patrascoiu (O.Patrascoiu@kent.ac.uk)

Description: The submission uses the terms of Attributes and Association Ends, which are no longer used in UML 2.0. In order to align OCL 2.0 and UML 2.0 specifications I think that the expression package should look like:



I also think that the OCL grammar should be rewritten accordingly.

Undefined values, isEmpty() and Collections

Author: Octavian Patrascoiu (O.Patrascoiu@kent.ac.uk)

Description: Most of the modern OO languages support null values, but OCL does not. In order to map null values into OCL concepts we used the undefined value. Unfortunately, OCL offers two choices to test if a value is undefined or not: isEmpty and oclIsUndefined. Using isEmpty for such a purpose is some how confusing:

the result of property->isEmpty() must be true if the value of the property null/undefined

the result of Set{1/0, 1/0}->isEmpty() must be false

These situations are a source of errors and confusion at the implementation level. I think that `isEmpty()` should be used only to test if a collection is empty or not; the undefined values should be tested using `ocIslUndefined`. This operation should be also valid on collections. This approach will also work nice and clear for nested collections.