$\mathbb{K}$ Semantics for OCL

a Proposal for a Formal Definition for OCL

Vlad Rusu, Inria Lille, France
Dorel Lucanu, Al. I. Cuza University of Iaşi, Romania

10/08/2011, 2nd $\mathbb{K}$ Workshop, Cheile Grădiştei

Outline

Outline

OCL

☺ OCL is a formal language used to describe expressions like constraints or queries over objects in a UML model

- the constraints are used to give an exact description of the information contained in the models
- the queries are used to analyse these models and to validate them

☺ the evaluation of the OCL expressions does not have side effects

☹ OCL is not yet widely adopted in industry

- the lack of proper and integrated tool support
- experience has shown that the language definition is not precise enough

OMG Standard

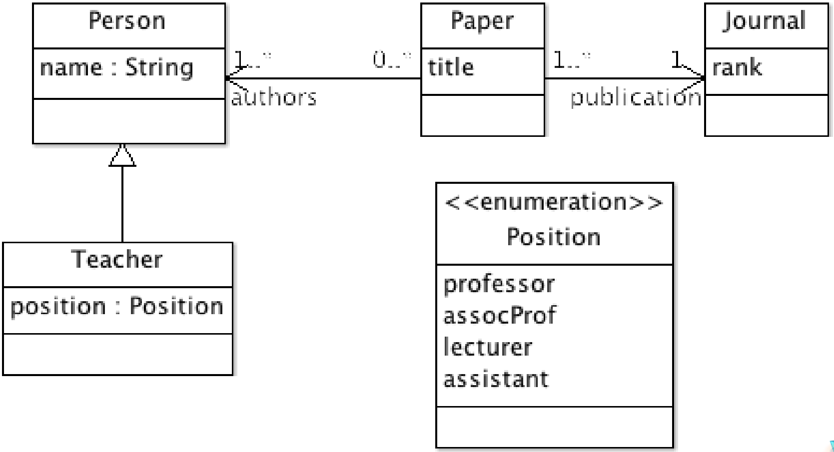
- an intuitive description of the OCL
- abstract syntax
  – as a MOF (Meta Object Facility) compliant metamodel
- concrete syntax
  – as a full attribute grammar
  – a map from abstract syntax to concrete one is given
- semantics described using UML
  – both semantic domains and evaluation are described as UML packages
  – the evaluation can be seen as a mapping between semantic domain and abstract syntax
- OCL standard library
  – predefined types, their operations, and predefined expression templates
- The Use of OCL Expressions in UML Models

OMG Standard is Not Quite Formal

- an informative formal semantics is given in Appendix A
- heavily based on mathematical notation
- without an execution engine is useless
- the relationship with the UML semantics not investigated
- each framework using OCL should have its own implementation
- no means to verify if an implementation conforms to the standard

Outline

Relation to the UML metamodel



Example of OCL Query

```
Paper.allInstances()->select(
        authors->includes("John")
)
```

All instances of the class Paper having "John" as author or co-author

Example of OCL Constraint

```
context Teacher inv:
        self.position == Professor implies
            Paper.allInstances()->select(
                  authors->includes(self.name)
            ).publication.rank->sum() >= 6.5
```

Any professor must have the sum of paper ranks at least 6.5, where the rank of a paper is given by the rank of the publication which it appears in.

Types in OCL

- basic types: Integer, Real, Boolean, String
- enumeration types
- collections: Set(T), OrderedSet(T), Bag(T), Sequence(T)
  Collection is the abstract supertype of all collection types
- structured values: Tuple(name:String, age:Integer)
- object types: each class c induces a type t_c having the same name as the class
  The domain of a class c is the set of objects that can be created by this class and all of its child classes

OCL Operations over Collections

collection->size() : Integer
collection->includes(object : T) : Boolean
collection->excludes(object : T) : Boolean
collection->count(object : T) : Integer
collection->includesAll(c2 : Collection(T)) : Boolean
collection->excludesAll(c2 : Collection(T)) : Boolean
collection->isEmpty() : Boolean
collection->notEmpty() : Boolean
collection->sum() : T
collection->product(c2: Collection(T2)) : Set(Tuple(first: T, second: T2))

OCL Collection Operations

collection->select(boolean-expression)
collection->reject(v : Type | boolean-expression-with-v)
collection->collect(v : Type | expression-with-v)
collection->forAll(v : Type | boolean-expression-with-v)
collection->exists(v : Type | boolean-expression-with-v)
collection->iterate(elem : Type; acc : Type = <expression> |
expression-with-elem-and-acc)

Object Models Syntax

an object model includes:

- a set of classes *CLASS*
- a set of attributes for each class ATT_c
- a set of operations for each class OP_c
- a set of associations with role names and multiplicities $ASSOC_c$
- a generalization hierarchy over classes $\prec$

System State

A system state for a model M is a structure $\sigma(M) = (\sigma_{CLASS}, \sigma_{ATT}, \sigma_{ASSOC})$, where

- $\sigma_{CLASS}(c)$ contain all objects of a class $c \in CLASS$ existing in the system state
- σ_{ATT} assigns attribute values to each object: $\sigma_{ATT}(a) : \sigma_{CLASS}(c) \to I(t)$ for each $a : t_c \to t \in ATT_c^*$
- σ_{ASSOC} contain links connecting objects: $\sigma_{ASSOC}(as) \subset I_{ASSOC}(as)$ for each $as \in ASSOC$

Environment

A context for evaluation is given by an environment $\pi = (\sigma, \beta)$ consisting of a system state σ and a variable assignment $\beta : Var_t \to I(t)$.

A system state σ provides access to the set of currently existing objects, their attribute values, and association links between objects.

A variable assignment β maps variable names to values.

Outline

1. Introduction & Motivation
2. OCL
3. $\mathbb{K}$ Semantics for OCL
4. Conclusion

Types (values.k) 1/2

– partial implemented
– we use the sort Val for OCL values
– OCL basic types Integer, Boolean, and String are described by the $\mathbb{K}$ types Int, Bool, and String, respectively

```
syntax Val ::= val ( Bag )
```

– if expressions are correctly typed, we may work with Collection instead of Collection(T)

```
syntax K ::= typedElt ( Val , Id )
```

– we extend $\mathbb{K}$ data structure Bag to implement OCL type Bag
– subtype relation Bag $\leq$ Val is an injection

```
syntax Val ::= val ( Bag )
```

Types (values.k) 2/2

– the relation Bag(T) $\leq$ Bag for scalar types T is modelled by the injection BagItem

– a scalar value v has two representations:
v and $val(BagItem(v))$

– the operations should be defined both representations of the scalars

```
rule KL:KLabel(val(BagItem(Val1:Val)),,val(BagItem(Val2:Val)))
     =>
     KL:KLabel(Val1,, Val2)
     if (binaryOp(KL))
     andBool isScalar((Val1))
     andBool isScalar((Val2))
     [structural]
```

'_+_(val(BagItem(3)),, val(BagItem(7))) => '_+_(3,, 7)

Model Syntax in 𝕂

```
kmod META-MODEL-INTERFACE is including K + #ID

   syntax K ::= Id

   syntax Bag ::= children ( Id )

   syntax Map ::= attributeDecl ( Id )

   syntax Bool ::= isEnum ( Id )

   syntax Bag ::= values ( Id )

endkm
```

Computations

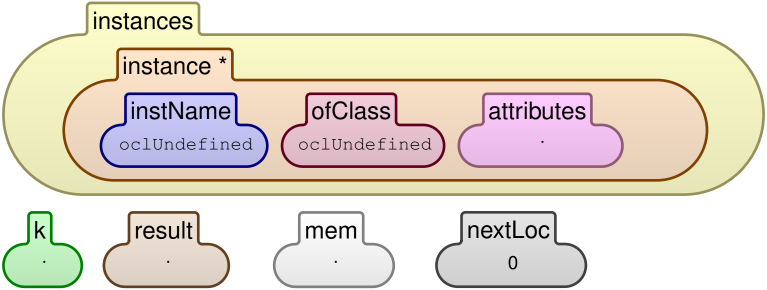
– strictness attributes

```
syntax Exp ::=  Exp ->forAll( Id  `| Exp ) [strict (1) prec 1]
              | Exp ->exists( Id  `| Exp ) [strict (1) prec 1]
              | Exp ->select( Id `| Exp ) [strict (1) prec 1]
              | Exp ->collect( Id  `| Exp )  [strict (1) prec 1]
              | let Id `=   Exp  in Exp endlet [strict(2)]
              | if Exp then Exp else Exp endif [strict(1)]
```

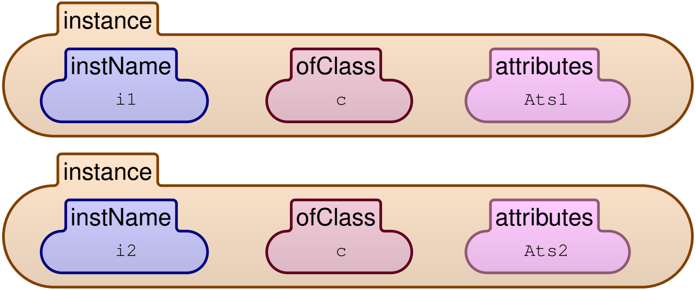
– the values (results)

```
syntax KResult ::= Val
syntax Exp ::= Val
syntax K ::= Exp
```

Configuration



Correspondece with the System State: $\sigma_{CLASS}(c)$



...

Correspondece with the System State: σ_{ATT}

$$\sigma_{ATT}(name)(pers) = "John"$$

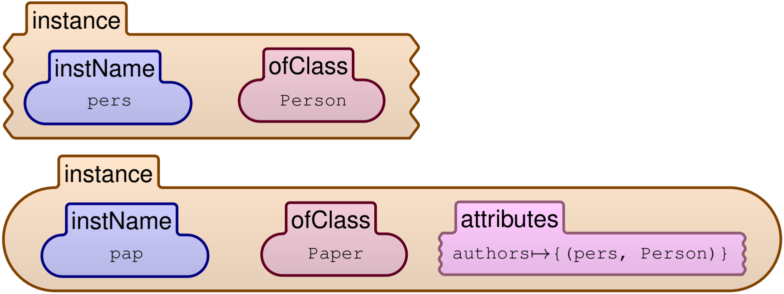
instName: `pers`

ofClass: `Person`

attributes: `name↦("John", String)`

Correspondece with the System State: σ_{ASSOC}

$$associates(as) = \{Paper, Person\}$$
$$roles(as) = \{authors\}$$
$$\sigma_{ASSOC}(authors)(pap) = \{pers\}$$



$\mathbb{K}$ Semantics for "forAll" operator 1/2

The base case is trivial:

k

$$\frac{\texttt{val}(\cdot)\ \texttt{->forAll}(\ \_\mathit{List}\{K\}\ \mid\ \_\mathit{List}\{K\}\ )}{\text{true}}$$

The inductive step uses the substitution operator:
$\mathit{Exp} ::= \mathit{Exp}\ [\ K\ /\ \mathit{Id}\ ]$ [ditto]

The K items "boxed" as bag items must be "unboxed":
$K ::= \texttt{open}(\ \mathit{BagItem}\ )$
$\texttt{open}(\texttt{BagItem}(\ K\ ))\ \Rightarrow\ K$

𝕂 Semantics for "forAll" operator 2/2

Now, the rule for the inductive step is written as:

k

$$\frac{\texttt{val}\,(\ \mathit{BgIt}\ \ \mathit{Bg}\ )\ \texttt{->forAll}\,(\ \mathit{Var}\ |\ \mathit{BEXP}\ )}{\texttt{if}\ \mathit{BEXP}\ [\ \texttt{open}\,(\ \mathit{BgIt}\ )\ /\ \mathit{Var}\ ]\ \texttt{then}\ \texttt{val}\,(\ \mathit{Bg}\ )\ \texttt{->forAll}\,(\ \mathit{Var}\ |\ \mathit{BEXP}\ )\ \texttt{else}\ \text{false}\ \texttt{endif}}$$

𝕂 Semantics for "_#_ " 1/5

NB. Correct syntax is ColExp.At but Maude parser ...

– returns the values of the attribute "At" for all objects obtained by the evaluation of "ColExp"

– the operator is strict in the first argument

– we need a way to store the results

... ihm, the evaluation of the argument "ColExp" could imply the evaluation other subexpressions _#_

$\mathbb{K}$ Semantics for "Exp # Id" 2/5

Our solution:

- use a memory cell mem which stores pairs of maps

$$location \mapsto sharpExpression$$
$$location + 1 \mapsto partialValue$$

- an operator * such that the evaluation of a sharp expression is replaced with the evaluation of $*\ location$
- the intermediate steps update the value from $location + 1$
- when this value (of the sharp (sub)expression from the *location*) is completely computed, it replaces $*\ location$ in the k cell.

𝕂 Semantics for "Exp # Id" 3/5

– additional syntax

$$K ::= \star\ Nat$$
$$\quad \mid\ [\ K\ ,\ K\ ]$$

– initial step

k

$$\frac{\texttt{val}(\ REMAINING\ )\ \#\ AT}{\star\ N}$$

mem

$$\frac{\cdot}{N \mapsto [\ \texttt{val}(\ REMAINING\ )\ \#\ AT\ ,\ N +_{Nat} 1\ ]\quad N +_{Nat} 1 \mapsto \texttt{val}(\ \cdot\ )}$$

nextLoc

$$\frac{N}{N +_{Nat} 2}$$

𝕂 Semantics for "Exp # Id" 4/5

– loop step

k

$\star\ N$

mem

$$N \mapsto [\ \frac{\texttt{val}(\ INAME : Cls\ \ REMAINING\ )\ \#\ AT}{\texttt{val}(\ REMAINING\ )\ \#\ AT}\ ,\ N'\ ]\ \ N' \mapsto \texttt{val}(\ \_Bag\ \frac{\cdot}{Bg}\ )$$

instances

instance

instName: $INAME$

ofClass: Cls

attributes: $AT \mapsto \texttt{typedElt}(\ \texttt{val}(\ Bg\ )\ ,\ T\ )$

𝕂 Semantics for "Exp # Id" 5/5

– last step

k

$$\frac{\star\ N}{\mathtt{val}(\ Bg\ )}$$

mem

$$\frac{N \mapsto [\ \mathtt{val}(\ \cdot\ )\ \#\ AT\ ,\ N'\ ]\quad N' \mapsto \mathtt{val}(\ Bg\ )}{\cdot}$$

Outline

Conclusion

- we presented a 𝕂 definition for a kernel of OCL
- the description in 𝕂 not trivial
- the use of substitution mechanism
- the mechanism for _#_
- it started as a component of the DSML project, but we think that OCL deserves to be completely developed as an independent project
- it could be the first standard formal semantics for OCL (it is a definition, not a implementation!)

Future work

- complete description of the types
- operations
- contexts
- missing OCL expressions
- relationship with the UML semantics
- online tool for experiments and testing bench