Introduction to OCL

Bernhard Beckert

Universität Koblenz-Landau
Object Constraint Language

- Part of the UML standard.
Object Constraint Language

- Part of the UML standard.

Object Constraint Language

- Part of the UML standard.
- (Quite) easy to read syntax.
Object Constraint Language

- Part of the UML standard.
- (Quite) easy to read syntax.
- Why? Because UML is not enough!
UML is not enough...

- Possible number of owners a car can have
- Required age of car owners
- Requirement that a person may own at most one black car
Some OCL examples I

“A vehicle owner must be at least 18 years old”: 
Some OCL examples I

“A vehicle owner must be at least 18 years old”:

context Vehicle
inv: self.owner.age ≥ 18
Some OCL examples I

“A vehicle owner must be at least 18 years old”:

context Vehicle
inv: self. owner. age >= 18
“A vehicle owner must be at least 18 years old”:

context Vehicle
inv: self.owner.age >= 18
“A vehicle owner must be at least 18 years old”:  

context Vehicle 
inv: self. owner. age >= 18
Some OCL examples I

“*A vehicle owner must be at least 18 years old*”:

```
context Vehicle
inv: self.owner.age >= 18
```
“A vehicle owner must be at least 18 years old”:

context Vehicle
inv: self.owner.age >= 18

What does this mean, instead?

context Person
inv: self.age >= 18
Some OCL examples

“A vehicle owner must be at least 18 years old”:
context Vehicle
inv: self.owner.age >= 18

“A car owner must be at least 18 years old”:
context Car
inv: self.owner.age >= 18
Some OCL examples II

```
Person

name: String
age: int

query
getName(): String
birthday()
setAge(newAge: int): int

Vehicle

colour: Colour

Car

Bike

<<enumeration>>

Colour

black(): Colour
white(): Colour
red(): Colour

“Nobody has more than 3 vehicles”: 

"All cars of a person are black": 
context Person
inv: self.eet –
forAll(v | v.colour = #black)

"Nobody has more than 3 black vehicles": 
context Person
inv: self.eet –
select(v | v.colour = #black) –
size < = 3
```
“Nobody has more than 3 vehicles”:

context Person
inv: self.fleet->size <= 3

or change multiplicity
Some OCL examples II

“All cars of a person are black”:
“All cars of a person are black”:

context Person
class self.fleet -> forAll(v | v.colour = #black)
“All cars of a person are black”:

```ocl
class Person

attribute name : String
attribute age : Int

<<query>>
getName() : String
birthday()
setAge(newAge : Int) : Int

context Person
inv: self.fleet -> forAll(v | v.colour = #black)
```

“Nobody has more than 3 black vehicles”:
“All cars of a person are black”:

context Person
inv: self.fleet→forall(v | v.colour = #black)

“Nobody has more than 3 black vehicles”:

context Person
inv: self.fleet→select(v | v.colour = #black)→size <= 3
What does this mean?

**context** Person

**inv:** self.fleet->iterate(v; acc:Integer=0 | if (v.colour=#black) then acc + 1 else acc endif) <=3
context Person
inv: age<18 implies self.fleet->forall(v | not voclIsKindOf(Car))
context Person
inv: age<18 implies self.fleet->forall(v | not v.oclIsKindOf(Car))

“A person younger than 18 owns no cars.”
Some OCL examples IV — oclIsKindOf

context Person
inv: age<18 implies self.fleet->forall(v | not v.oclIsKindOf(Car))

“A person younger than 18 owns no cars.”

“self” can be omitted.
Some OCL examples IV — oclIsKindOf

context Person
inv: age<18 implies self.fleet→forall(v | not v.oclIsKindOf(Car))

“A person younger than 18 owns no cars.”

“self” can be omitted.

Logical Junctors: and, or, not, implies, if...then...else...endif, =
Some OCL examples V — allInstances

context Car
inv: Car.allInstances() -> exists(c | c.colour=#red)
Some OCL examples V — allInstances

context Car
inv: Car.allInstances() -> exists(c | c.colour=#red)

“There is a red car.”
OCL pre-/post conditions — Examples

So far only considered class invariants.
OCL pre-/post conditions — Examples

So far only considered class invariants.

OCL can also specify operations:
So far only considered class invariants.

OCL can also specify operations:

“If setAge(…) is called with a non-negative argument then the argument becomes the new value of the attribute age.”

context Person::setAge(newAge:int)
pre: newAge >= 0
post: self.age = newAge
OCL pre-/post conditions — Examples

So far only considered class invariants.

OCL can also specify operations:

“Calling birthday() increments the age of a person by 1.”

```plaintext
context Person::birthday()
post: self.age = self.age@pre + 1
```
OCL pre-/post conditions — Examples

So far only considered class invariants.

OCL can also specify operations:

“Calling getName() delivers the value of the attribute name.”

context Person::getName()
post: result = name
Queries

Special to OCL are operations with a \textit{<<query>>} stereotype:

\textbf{Only these} operations can be used within an OCL expression.
Special to OCL are operations with a «query» stereotype:

Only these operations can be used within an OCL expression.

“Calling getName() delivers the value of the attribute name.”

class Person

context Person
inv: self.getName() = name
OCL Basics

- OCL is used to specify **invariants** of objects and **pre- and post conditions** of operations. Makes UML (class) diagrams more precise.
OCL Basics

- OCL is used to specify invariants of objects and pre- and post conditions of operations. Makes UML (class) diagrams more precise.
- OCL expressions use vocabulary of UML class diagram.
OCL Basics

- OCL is used to specify invariants of objects and pre- and post conditions of operations. Makes UML (class) diagrams more precise.

- OCL expressions use vocabulary of UML class diagram.

- OCL attribute accesses “navigate” through UML class diagram.
OCL Basics

- OCL is used to specify **invariants** of objects and **pre- and post conditions** of operations. Makes UML (class) diagrams more precise.

- OCL expressions use vocabulary of UML class diagram.

- OCL attribute accesses “navigate” through UML class diagram.

- “context” specifies about which elements we are talking.
OCL Basics

- OCL is used to specify **invariants** of objects and **pre- and post conditions** of operations. Makes UML (class) diagrams more precise.

- OCL expressions use vocabulary of UML class diagram.

- OCL attribute accesses “navigate” through UML class diagram.

- “context” specifies about which elements we are talking.

- “self” indicates the current object. “result” the return value.
OCL Basics (cont.)

- OCL can talk about collections (here: sets).

  Operations on collections: →

  Example operations: select, forAll, iterate
OCL Basics (cont.)

- OCL can talk about collections (here: sets).

  Operations on collections: –>

  Example operations: select, forAll, iterate

- “iterate” can simulate all other operations on collections.
OCL can talk about collections (here: sets).

Operations on collections: –>

Example operations: select, forAll, iterate

“iterate” can simulate all other operations on collections.

Queries (= side-effect-free operations) can be used in OCL expressions.
TogetherCC cannot process OCL constraints. It is however possible to specify textual invariants and pre- and post conditions.

With the KeY extensions to TogetherCC syntax (type) checks of OCL constraints are possible.
System state
System state

id0815:Person
  name = "Jane"
  age = 5

harley17:Bike
  colour = idBlack

idBlack:Colour
  black() = idBlack
  white() = idWhite
  red() = idRed

id0825:Person
  name = "Paul"
  age = 25

bmw3:Car
  colour = idWhite

idWhite:Colour
  black() = idBlack
  white() = idWhite
  red() = idRed

idRed:Colour
  black() = idBlack
  white() = idWhite
  red() = idRed

ownership

context Vehicle

inv: self.owner.age = 18

context Person

inv: self.eet–forAll(v | v.colour = #black)

4

context Person

inv: self.eet–select(v | v.colour = #black)–size < 3

4

inv: Car.allInstances()–exists(c | c.colour=#red)

4
context Vehicle
inv: self.owner.age >= 18
context Vehicle
inv: self.owner.age >= 18

System state

id0815:Person
  name = "Jane"
  age = 5

idBlack:Colour
  black() = idBlack
  white() = idWhite
  red() = idRed

id0825:Person
  name = "Paul"
  age = 25

bmw3:Car
  colour = idWhite

harley17:Bike
  colour = idBlack

idWhite:Colour
  black() = idBlack
  white() = idWhite
  red() = idRed

idRed:Colour
  black() = idBlack
  white() = idWhite
  red() = idRed

context Person
inv: self.name -
  forAll(v | v.colour = #black)

context Person
inv: self.eet-
  size < 3

context Car
inv: Car.allInstances()
  exists(c | c.colour=#red)
context Vehicle
inv: self.owner.age >= 18

context Person
inv: self.fleet->forall(v | v.colour = #black)
context Vehicle
inv: self.owner.age >= 18

context Person
inv: self.fleet->forAll(v | v.colour = #black)
context Vehicle
inv: self.owner.age >= 18 ✓

context Person
inv: self.fleet->forAll(v | v.colour = #black) ×

context Person
inv: self.fleet->select(v | v.colour = #black)->size <= 3
context Vehicle
def: self.owner.age >= 18

context Person
def: self.fleet -> forAll(v | v.colour = #black)

context Person
def: self.fleet -> select(v | v.colour = #black) -> size <= 3
context Vehicle
inv: self.owner.age >= 18

context Person
inv: self.fleet->forall(v | v.colour = #black) 

context Person
inv: self.fleet->select(v | v.colour = #black)->size <= 3
inv: Car.allInstances()->exists(c | c.colour=#red)
context Vehicle
inv: self.owner.age >= 18

context Person
inv: self.fleet->forAll(v | v.colour = #black)

context Person
inv: self.fleet->select(v | v.colour = #black)->size <= 3
inv: Car.allInstances()->exists(c | c.colour=#red)