Introduction to OCL

Andreas Roth
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...

How many persons can own a car?

How old must a car owner be?

How can we require that a person must at most own one black car?
Some OCL examples I

```
Person
name: String
age: Integer

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

Vehicle
colour: Colour

enumeration Colour
#black
#white
#red

Car
Bike

context Vehicle
inv: self.owner.age > 18

context Car
inv: self.owner.age > 18
```

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

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

class Vehicle

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

```
Person

name: String
age: Integer

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

Vehicle

colour: Colour

Car

Bike

<<enumeration>>

Colour

#black
#white
#red

"A vehicle owner must be at least 18 years old":

countext 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
Some OCL examples I

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

```ocl
class Vehicle {
  owner: Person
  age: Integer

  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
```

Praktikum WS0304: Formale Entwicklung objektorientierter Software — Introduction to OCL – p.4
Some OCL examples I

"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 I

"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
“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
“Nobody has more than 3 vehicles”: 
Some OCL examples II

```
Person
name: String
age: Integer

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

Vehicle
colour: Colour

Car
Bike

<<enumeration>>
Colour
#black
#white
#red

```

“Nobody has more than 3 vehicles”:

```
context Person
inv: self.fleet->size <= 3
```
Some OCL examples II

```
Person

name:String
age:Integer

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

 ENUMERATION

Colour
#black
#white
#red

Vehicle

colour:Colour

Car
Bike

1 0..*

owner

ownership

fleet

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

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

\[
\text{context Person} \\
\text{inv: } \text{self.fleet} \rightarrow \text{forall}(v \mid v.\text{colour} = \#\text{black})
\]

“Nobody has more than 3 black vehicles”: 
Some OCL examples II

“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
Some OCL examples III — iterate

What does it mean?
context Person
inv: self.fleet->iterate(v; acc:Integer=0 | if (v.colour=#black) then acc + 1 else acc endif) <=3
Some OCL examples IV — oclIsKindOf

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

```ocl
class Person
  name : String
  age : Integer

  operation getName() : String
  operation birthday() : Integer
  operation setAge(newAge : Integer) : Integer

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

Person
- name: String
- age: Integer

\(<\text{query}>\)
- getName(): String
- birthday()
- setAge(newAge: Integer): Integer

Vehicle
- colour: Colour

\(<\text{enumeration}>\)
- Colour
  - #black
  - #white
  - #red

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 not 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.”

context Person::birthday()
post: self.age = self.age@pre + 1
So far only considered class invariants.

OCL can also specify operations:

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

```ocl
class Person {
  name: String
  age: Integer
  
  operation getName(): String
  operation birthday(): String
  operation setAge(newAge: Integer): Integer
}

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

Special to OCL are operations with a "query" stereotype:

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.”

```java
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 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.

Queries (= side effect free operations) can be used in OCL expressions.
TogetherCC itself 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

id0815: Person
  name = Paulchen
  age = 5

id0825: Person
  name = Paul
  age = 25

idhd135: Bike
  colour = #black

mb374: Car
  colour = #white

idb: Colour
  value = #black

idw: Colour
  value = #white

idr: Colour
  value = #red

ownership

context Vehicle
  inv: self.owner.age > 18

context Person
  inv: self.fleet.forAll(v | v.colour = #black).size < 3
  inv: Car.allInstances().exists(c | c.colour = #red)
System state

(depicted by a UML object diagram)

id0815:Person
name = Paulchen
age = 5

id0825:Person
name = Paul
age = 25

idhd135:Bike
colour = #black

mb374:Car
colour = #white

idb:Colour
value = #black

idw:Colour
value = #white

idr:Colour
value = #red

inv: self.owner.age = 18
inv: self.fleet–forAll(v | v.colour = #black)–size < 3
inv: Car.allInstances()–exists(c | c.colour=#red)
System state

(depicted by a UML object diagram)

context Vehicle
inv: self.owner.age ≥ 18
context Vehicle
inv: self.owner.age >= 18 ✓
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)
System state

(depicted by a UML object diagram)

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
System state

(depicted by a UML object diagram)

** invo: self.owner.age ≥ 18 ✓**

** invo: self.fleet→forall(v | v.colour = #black) ❌**

** invo: self.fleet→select(v | v.colour = #black)→size ≤ 3 ✓**
System state

(depicted by a UML object diagram)

<table>
<thead>
<tr>
<th>id0815:Person</th>
<th>idhd135:Bike</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = Paulchen</td>
<td>colour = #black</td>
</tr>
<tr>
<td>age = 5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id0825:Person</th>
<th>mb374:Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>name = Paul</td>
<td>colour = #white</td>
</tr>
<tr>
<td>age = 25</td>
<td></td>
</tr>
</tbody>
</table>

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)
System state

(idepicted by a UML object diagram)

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 Person::getName()
post: result = name
System State

(depicted by a UML object diagram)

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

Praktikum WS0304: Formale Entwicklung objektorientierter Software — Introduction to OCL – p.15
System State

Given a UML class diagram, a system state (snapshot) is defined by

- a UML object diagram (for the class diagram), giving
System State

Given a UML class diagram, a system state (snapshot) is defined by

- a UML object diagram (for the class diagram), giving
  - the set of existing instances,
Given a UML class diagram, a system state (snapshot) is defined by

- a UML object diagram (for the class diagram), giving
  - the set of existing instances,
  - attribute-value-assignments
System State

Given a UML class diagram, a system state (snapshot) is defined by

- a UML object diagram (for the class diagram), giving
  - the set of existing instances,
  - attribute-value-assignments
  - instances of associations ("links")
System State

Given a UML class diagram, a system state (snapshot) is defined by

- a UML object diagram (for the class diagram), giving
  - the set of existing instances,
  - attribute-value-assignments
  - instances of associations ("links")
- an interpretation for operations,
System State

Given a UML class diagram, a system state (snapshot) is defined by

- a UML object diagram (for the class diagram), giving
  - the set of existing instances,
  - attribute-value-assignments
  - instances of associations ("links")
- an interpretation for operations,
- (standard) interpretation for predefined primitive data types
  (e.g. Integer, String, . . .)
OCL Constraints are satisfied by certain system states.
OCL Constraints are satisfied by certain system states.

Given an implementation of a class diagram, a sequence of system states is reached.
OCL Constraints are satisfied by certain system states.

Given an implementation of a class diagram, a sequence of system states is reached.

The interesting question is: How can we check that constraints are satisfied in all system states that are reached by an implementation?
OCL Constraints are satisfied by certain system states.

Given an implementation of a class diagram, a sequence of system states is reached.

The interesting question is: How can we check that constraints are satisfied in all system states that are reached by an implementation?

Answer in three weeks.
P. Schmitt:

Skript ”Formale Spezifikationssprachen”

Jos Warmer and Anneke Kleppe:

The Object Constraint Language: Precise Modelling with UML.

UML 1.5 OCL Specification.


UML 2.0 OCL Revised submission to OMG.