

Formale Modellierung  
Vorlesung vom 31.05.12: OCL — Die Object Constraint Language

Till Mossakowski & Christoph Lüth

Universität Bremen

Sommersemester 2012

Rev. 1713

1 [12]

## OCL

- ▶ Object Constraint Language
- ▶ Mathematisch **präzise** Sprache für UML
- ▶ OO meets Z
- ▶ Entwickelt in den 90ern
- ▶ Formale **Constraints** an UML-Diagrammen

2 [12]

## OCL Basics

- ▶ **Getypte** Sprache
- ▶ Dreiwertige Logik (**Kleene-Logik**)
- ▶ Ausdrücke immer im **Kontext**:
  - ▶ **Invarianten** an Klassen, Interfaces, Typen
  - ▶ **Vor/Nachbedingungen** an Operationen oder Methoden

3 [12]

## OCL Syntax

- ▶ Invarianten:

```
context class
  inv: expr
```
- ▶ Vor/Nachbedingungen:

```
context Type :: op(arg1 : Type) : ReturnType
  pre: expr
  post: expr
```
- ▶ `expr` ist ein OCL-Ausdruck vom Typ `Boolean`

4 [12]

## OCL Typen

- ▶ Basistypen:
  - ▶ `Boolean`, `Integer`, `Real`, `String`
  - ▶ `OclAny`, `OclType`, `OclVoid`
- ▶ Collection types: `Set`, `OrderedSet`, `Bag`, `Sequences`
- ▶ Modelltypen

5 [12]

## Basistypen und Operationen

- ▶ `Integer` ( $\mathbb{Z}$ ) → OCL-Std. §11.5.2
- ▶ `Real` ( $\mathbb{R}$ ) → OCL-Std. §11.5.1
  - ▶ `Integer` Subklasse von `Real`
  - ▶ `round`, `floor` von `Real` nach `Integer`
- ▶ `String` (Zeichenketten) → OCL-Std. §11.5.3
  - ▶ `substring`, `toReal`, `toInteger`, `characters` etc.
- ▶ `Boolean` (Wahrheitswerte) → OCL-Std. §11.5.4
  - ▶ `or`, `xor`, `and`, `implies`
  - ▶ Sowie Relationen auf `Real`, `Integer`, `String`

6 [12]

## Collection Types

- ▶ `Set`, `OrderedSet`, `Bag`, `Sequence`
- ▶ Operationen auf allen Kollektionen: → OCL-Std. §11.7.1
  - ▶ `size`, `includes`, `count`, `isEmpty`, `flatten`
  - ▶ Kollektionen werden immer flachgeklopft
- ▶ `Set` → OCL-Std. §11.7.2
  - ▶ `union`, `intersection`,
- ▶ `Bag` → OCL-Std. §11.7.3
  - ▶ `union`, `intersection`, `count`
- ▶ `Sequence` → OCL-Std. §11.7.4
  - ▶ `first`, `last`, `reverse`, `prepend`, `append`

7 [12]

## Collection Types: Iteratoren

- ▶ Iteratoren: Funktionen höherer Ordnung
- ▶ Alle definiert über `iterate` → OCL-Std. §7.7.6:

```
coll-> iterate(elem: Type, acc: Type= expr | expr[elem, acc])

iterate(e: T, acc: T= v)
{
  acc= v;
  for (Enumeration e= c.elements(); e.hasMoreElements();){
    e= e.nextElement();
    acc.add(expr[e, acc]); // acc= expr[e, acc]
  }
  return acc;
}
```
- ▶ Iteratoren sind alle **strikt**

8 [12]

## Modelltypen

- ▶ Aus Attribute, Operationen, Assoziationen des Modells
- ▶ **Navigation** entlang der Assoziationen
- ▶ Für Kardinalität 1 Typ T, sonst Set(T)
- ▶ Benutzerdefinierte Operationen in Ausdrücken müssen zustandsfrei sein (Stereotyp <<query>>)

9 [12]

## Undefiniertheit in OCL

- ▶ Undefiniertheit **propagiert** (alle Operationen **strikt**) → OCL-Std. §7.5.11
- ▶ Ausnahmen:
  - ▶ Boolesche Operatoren (and, or **beidseitig** nicht-strikt)
  - ▶ Fallunterscheidung
  - ▶ Test auf Definiertheit: oclIsUndefined mit
$$\text{oclIsUndefined}(e) = \begin{cases} \text{true} & e = \perp \\ \text{false} & \text{otherwise} \end{cases}$$
- ▶ Resultierende Logik: **dreiwertig** (Kleene-Logik)
- ▶ Iteratoren: "semi-strikt"

10 [12]

## Style Guide

- ▶ Komplexe Navigation vermeiden ("Loose coupling")
- ▶ Adäquaten Kontext auswählen
- ▶ "Use of **allInstances** is discouraged"
- ▶ Invarianten aufspalten
- ▶ Hilfsoperationen definieren

11 [12]

## Zusammenfassung

- ▶ OCL erlaubt **Einschränkungen** auf Modellen
- ▶ Erlaubt **mathematisch** präzisere Modellierung
- ▶ Frage:
  - ▶ Werkzeugunterstützung?
  - ▶ Ziel: Beweise, Codegenerierung, ...?

12 [12]