Seminarvortrag zum Thema

OCL – The Object Constraint Language

Tobias Friedrich

19.07.2000

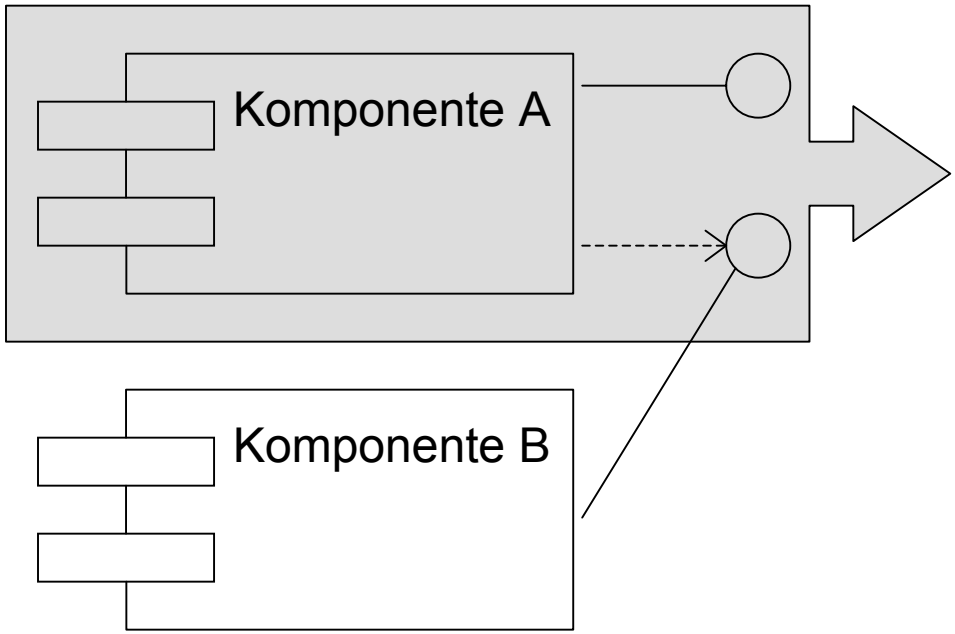
Gliederung

- Einleitung / Motivation
- Anwendung von OCL
- Bewertung

Einleitung / Motivation

- Zentrale Aspekte der komponentenbasierten Softwareentwicklung (CBD)
- *Design by Contract*
- Die Rolle formaler Methoden im CBD
- Was ist OCL?

Zentrale Aspekte des CBD (1)



- Bereitgestellte Schnittstellen
- Benötigte Schnittstellen
- Spezifikation des benötigten und bereitgestellten externen Verhaltens

Schlüsseltechnik bei der Entwicklung von "Komponenten-baukästen":

Festlegung präziser Schnittstellen

Zentrale Aspekte des CBD (2)

Besonderheiten bezüglich der Schnittstellen:

- Jede Komponente kommuniziert über ihre Schnittstellen mit anderen, in der Regel unbekannten Komponenten
- Listen von Methodensignaturen lassen keine Aussage über das erwartete bzw. zu erwartende Verhalten zu

⇨ Notwendigkeit präziser Schnittstellenbeschreibungen in Form eines Kontrakts

Design by Contract (1)

	Obligations	Benefits
Client	(Must ensure precondition) Be at the Santa Barbara airport at least 5 minutes before scheduled departure time. Bring only acceptable baggage. Pay ticket price.	(May benefit from postcondition) Reach Chicago.
Supplier	(Must ensure postcondition) Bring customer to Chicago.	(May assume precondition) No need to carry passenger who is late, has unacceptable baggage, or has not paid ticket price.

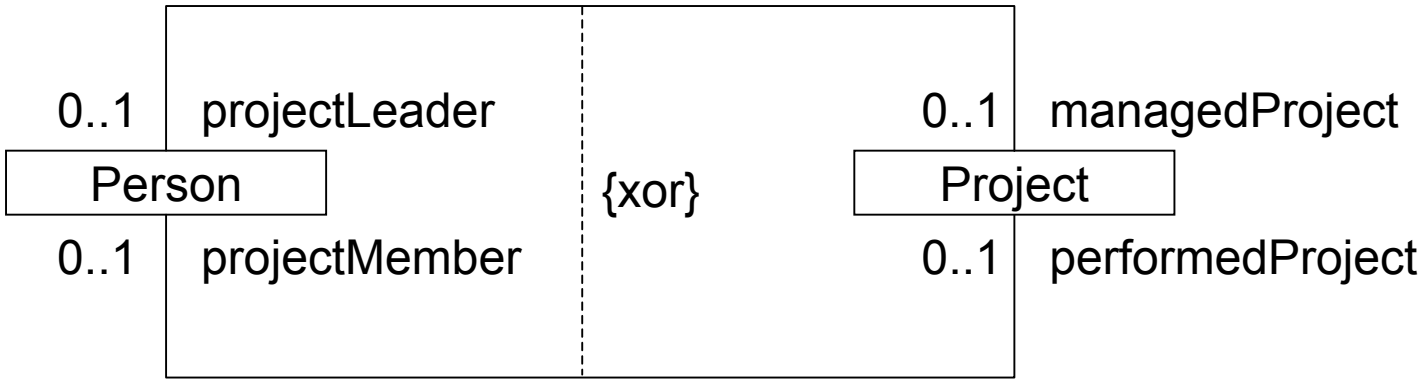
Design by Contract (2)

- Kontrakt = exakte Spezifikation der Schnittstelle eines Objekts, die Aussagen über das Verhalten der bereitgestellten Dienste macht
- Speziell für jeden Dienst:
  - Bedingungen, unter denen Dienst bereitgestellt wird (*Preconditions*)
  - Spezifikation des Ergebnisses, das der Dienst liefert, sofern Vorbedingungen eingehalten werden (*Postconditions*)
- Klare und eindeutige Festlegung von Verantwortlichkeiten

Die Rolle formaler Methoden im CBD (1)

- Erlauben eindeutige Beschreibung von Rechten und Pflichten innerhalb eines Kontrakts
- Graphische Notationen nicht immer präzise genug, um eindeutig die Spezifikation zu erfassen
- Eingeschränkte Ausdrucksstärke graphischer Notationen verhindert häufig, alle Aspekte der Spezifikation zu erfassen
- Zusätzliche Anforderungen an Modelle in natürlicher Sprache meist mehrdeutig

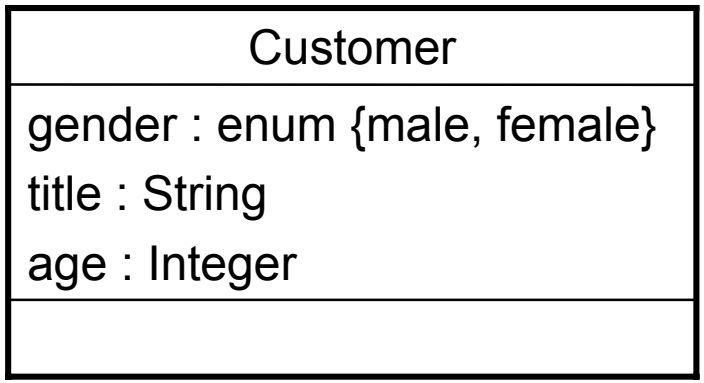
Die Rolle formaler Methoden im CBD (2)



Interpretationsmöglichkeiten:

- Eine Person managt entweder ein Projekt oder arbeitet an einem Projekt
- Ein Projekt wird entweder von einer Person gemanagt oder von einer Person durchgeführt

Die Rolle formaler Methoden im CBD (3)



Zusätzliche Anforderungen:

- Jeder Kunde muß mind. 18 Jahre alt sein
- Die Anrede männlicher Kunden ist "Mr.", die weiblicher Kunden "Mrs."

Was ist OCL? (1)

- Ursprünglich von IBM entwickelt (1995)
- Heute Bestandteil von UML
- Formale Sprache typisierter Ausdrücke
- Deklarative Sprache, d.h. Ausdrücke haben keine Nebeneffekte
- Modellierungs- und Spezifikationssprache, d.h. Implementierungsaspekte können in OCL nicht formuliert werden
- Ausgerichtet auf einfache Lesbarkeit und Handhabbarkeit (dennoch: basiert auf Mengentheorie und Prädikatenlogik)

Was ist OCL? (2)

- Sprache, mit der Zusicherungen (*Constraints*) für OO-Modelle formuliert werden können
- Constraints sind gültige OCL-Ausdrücke vom Typ Boolean; dazu zählen
  - Pre- und Postconditions
  - Invarianten
  - (Guards)
- Kann verwendet werden zum Festlegen von
  - Wohlgeformtheitseigenschaften von Metaclasses
  - anwendungsspezifischen Zusicherungen in UML-Modellen

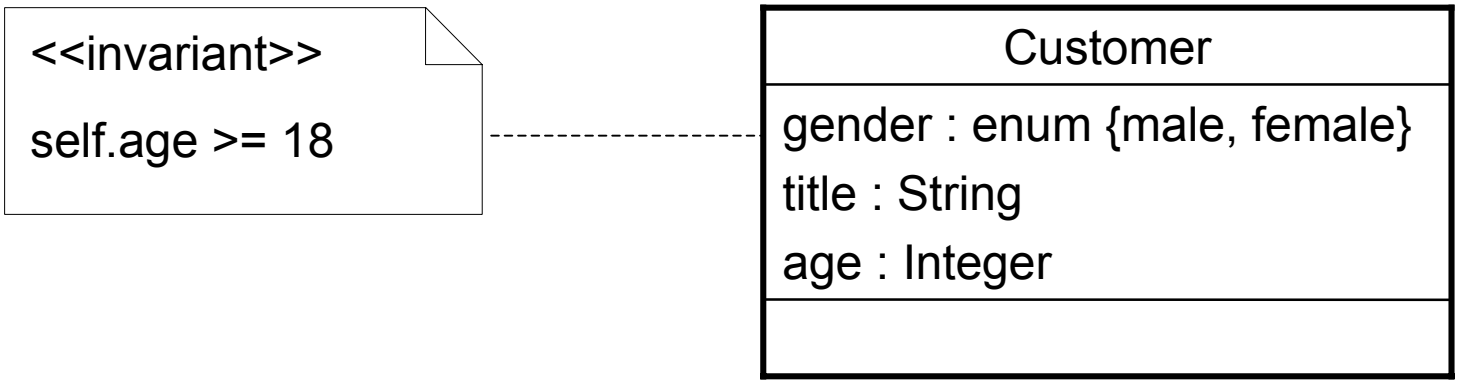
Anwendung von OCL

- Typen in OCL
- Kontext von OCL-Ausdrücken
- Konstrukte für Postconditions
- Collections und das Navigationsprinzip
- Operationen auf Collections
- Interfaces und Constraints

Typen in OCL

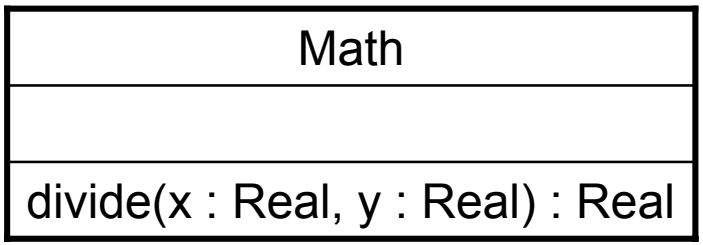
- Vordefinierte Typen
  - Basistypen: Integer, Real, String und Boolean
  - Collection-Typen: Collection, Set, Bag, Sequence
- (Meta-Typen: erlauben Zugriff auf das Metamodel)
- Benutzerdefinierte Typen
  - Enumeration
  - Alle Klassen, Typen und Interfaces aus dem Benutzermodell

Kontext von OCL-Ausdrücken (1)



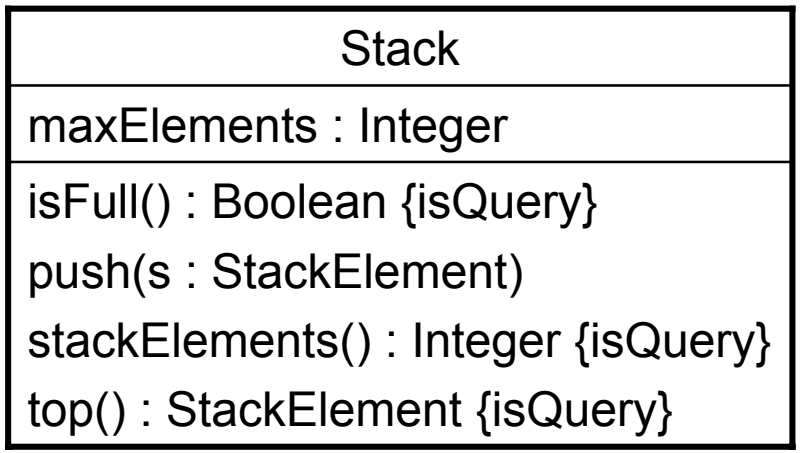

```
context Customer inv:
   (self.gender = #male implies self.title = 'Mr.') and
   (self.gender = #female implies self.title = 'Mrs.')

context c : Customer inv equivalentSample:
   c.title =
        if c.gender = #male then 'Mr.' else 'Mrs.' endif
```

Kontext von OCL-Ausdrücken (2)



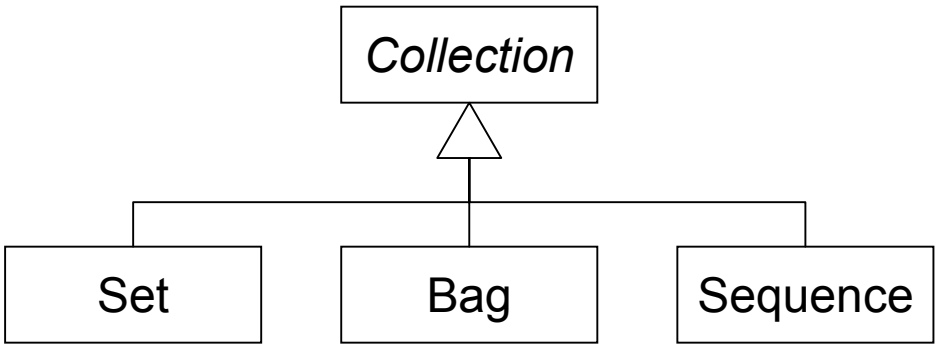

```
context Math::divide(x : Real, y : Real) : Real
   -- y is the divisor, x is the dividend
   pre : y <> 0.0
   post: result = x / y
```

Konstrukte für Postconditions



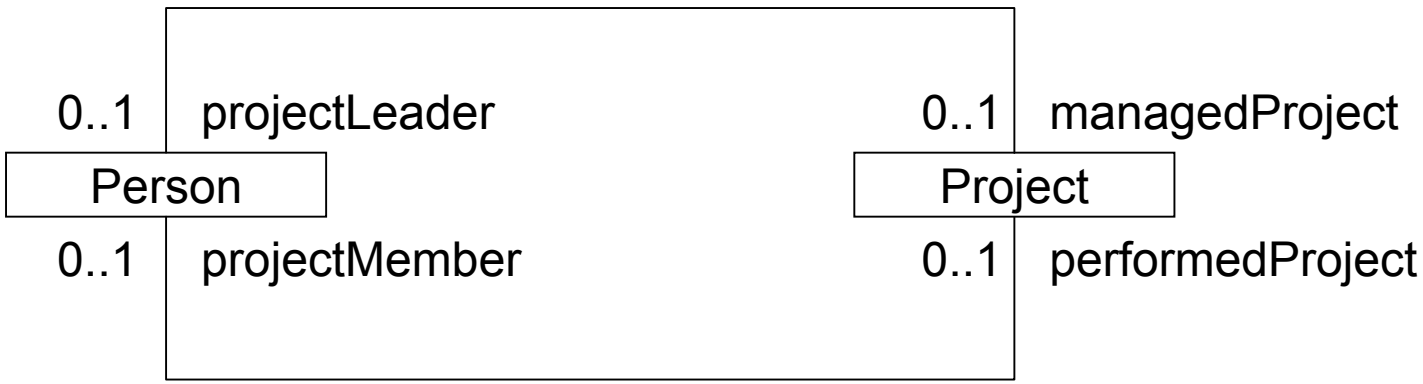

```
context Stack::isFull() : Boolean
pre : --
post: result = (stackElements() = maxElements)

context Stack::push(s : StackElement)
pre : not isFull()
post: stackElements() = stackElements@pre() + 1
      and top() = s
```

Collections und das Navigationsprinzip (1)



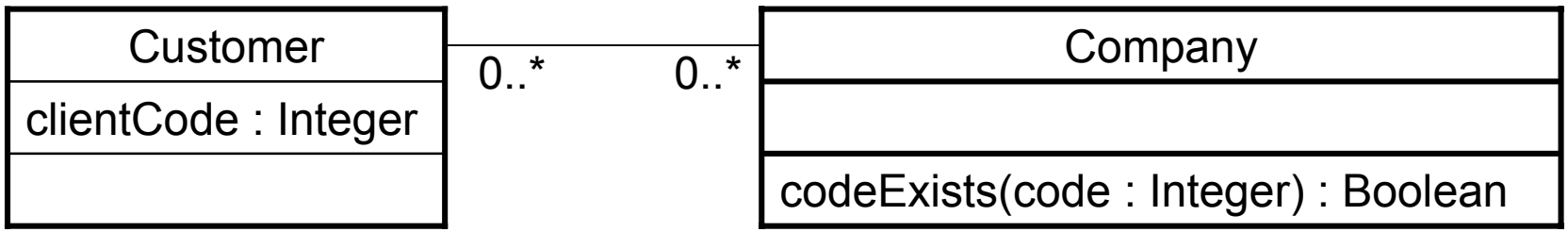
- Collection = Menge von Instanzen eines OCL-Typs
- Set: Collection ohne doppelte Elemente
- Bag: wie Set, kann aber doppelte Elemente enthalten
- Sequence: wie Bag, aber Elemente sind geordnet (nicht sortiert), d.h. jedes Element hat eine eindeutige Nummer

Collections und das Navigationsprinzip (2)




```
context Person inv:
   managedProject->isEmpty or
   performedProject->isEmpty
```

Operationen auf Collections




```
context Company inv:
   customer->forAll(c1, c2 : Customer | c1 <> c2
              implies c1.clientCode <> c2.clientCode)

context Company::codeExists(code : Integer) : Boolean
post: result =
  customer->select(c | c.clientCode = code)->size > 0
```

Interfaces und Constraints (1)

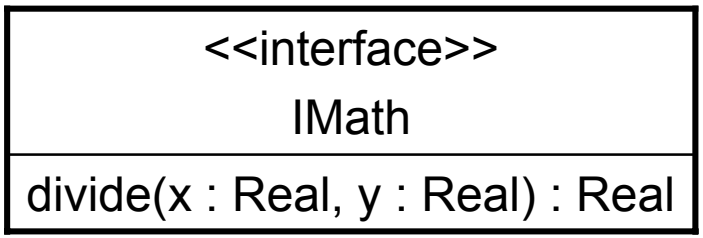
Besonderheiten von Interfaces in UML:

- enthalten keine Attribute, sondern nur Methoden(signaturen)
- haben keine Assoziationen zu anderen Klassen

⇨ Eingeschränkte Verwendung von OCL-Constraints auf Interface-Ebene:

nur das "beobachtbare Verhalten" kann spezifiziert werden

Interfaces und Constraints (2)



Verhalten von *divide* nur abhängig von:

- Parametern *x* und *y*

Verhalten von *divide* nicht abhängig von:

- internem Zustand von *IMath*
- Assoziationen zu anderen Klassen

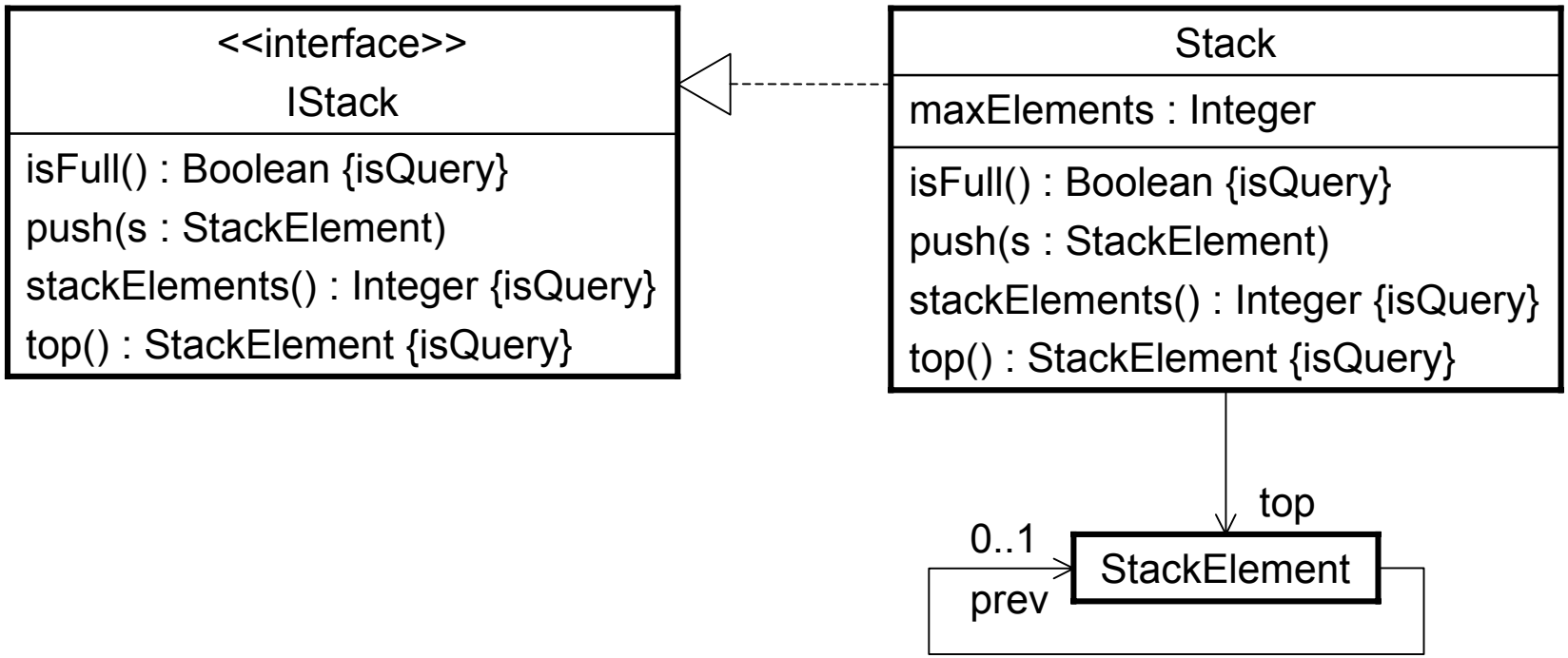
Interfaces und Constraints (3)

Zur Spezifikation von komplexeren Methoden, die

- mehr tun, als Eingabe- in Ausgabewerte zu transformieren
- auf internen Zuständen und/oder Assoziationen zu anderen Klassen basieren

ist die Offenlegung des Modells einer möglichen Implementierung notwendig, das eine vollständige Spezifikation dieser Methoden erlaubt.

Interfaces und Constraints (4)




```
context Stack::push(s : StackElement)
pre : not isFull()
post: stackElements() = stackElements@pre() + 1
      and top = s and top@pre = top.prev
```

Bewertung (1)

Vorteile von OCL:

- Präzision und Eindeutigkeit
- Bessere Dokumentation
- Kommunikation ohne Mißverständnisse
- Erweiterbarkeit

Bewertung (2)

Nachteile von OCL:

- Teilweise schwer lesbar und/oder erlernbar
- Teilweise unvollständige oder mehrdeutige Spezifikation der Semantik von OCL
- (Bisher) mangelhafte Integration in CASE-Tools (obwohl eigenständige Parser bereits verfügbar sind)