

## 4.7 Zusammenfassung und Ausblick

## 4.7 Zusammenfassung und Ausblick

- Forschungsgegenstände
  - Systematische Ermittlung der minimalen Menge der generierenden Operationen
  - Ermittlung der nötigen Axiome für die gewünschte / beabsichtigte Spezifikation

Anhaltspunkt:

$oi_1, \dots, oi_n$  als Menge der inspizierenden Operationen

$og_1, \dots, og_m$  als Menge der generierenden Operationen

Axiome:  $oi_i(og_j(\dots)) = \dots \quad \forall i = 1, \dots, n \quad \forall j = 1, \dots, m$

- Automatische Identifikation von Widersprüchen
- Automatische Identifikation von Redundanz
- Von der Spezifikation zum Programm ??

## 4.7 Zusammenfassung und Ausblick

- Vorteile der algebraischen Spezifikation
  - Spezifikation weitestgehend implementierungsunabhängig
  - Vollständig formale Spezifikation, daher für Verifikation und automatische Werkzeuge einsetzbar
  - Theoretisch fundiert
  - Kann auf abstrakte Datenobjekte, Datentypen und Klassen angewandt werden
  - Wirkung von Zugriffsoperationen ergibt sich durch wechselseitige statische Abhängigkeiten, nicht durch isolierte Betrachtung
  - Mit einiger Erfahrung kann man algebraische Spezifikationen lesen
  - Leicht erweiterbar, da zusätzliche Operationen nur jeweils eine neue Syntaxregel sowie zusätzliche Axiome aufbauend auf Grundoperationen erfordern

## 4.7 Zusammenfassung und Ausblick

- Nachteile der algebraischen Spezifikation
    - Konstruktion algebraischer Spezifikationen nicht trivial
    - Fälle wie z.B. Grenzbedingungen können leicht übersehen werden
    - Schwierig auf Konsistenz und Vollständigkeit zu prüfen
    - Operationen mit mehreren Wertebereichen sind schwierig zu beschreiben
- In der Praxis hat sich die algebraische Spezifikation nicht durchgesetzt!

## 4.7 Zusammenfassung und Ausblick

- Warum algebraische Spezifikation, modellbasierte Spezifikation und formale Spezifikation im allgemeinen?
  - Der Spezifizierer muss Basistechniken und Sprachen kennen
  - Formale Spezifikation sind eine bedeutsame Klasse in der Menge aller Spezifikationssprachen
  - Man muss wissen, was man mit formaler Spezifikation erreichen kann UND was nicht!

## 4.8 Literatur


- Ehrig, Mahr: Fundamentals of algebraic Specification 1: Equations and Initial Semantics, Springer Verlag, 1985
- Ehrig, Mahr: Fundamentals of algebraic Specification 2: Module Specifications and Constraints, Springer Verlag, 1990
- Ehrig, Mahr, Cornelius et.al.: Mathematisch-strukturelle Grundlagen der Informatik, Springer Verlag, 1999
- Ehrich, Gogolla, Lipeck: Algebraische Spezifikation abstrakter Datentypen, Teubner Verlag, 1989

UNIVERSITÄT LEIPZIG

LPZ  E-BUSINESS  
*applied telematics*

Object-Z

UNIVERSITÄT LEIPZIG

LPZ  E-BUSINESS  
*applied telematics*

## 5.1 Was ist Object-Z?

- Object-Z
  - Klassenbasierter Ansatz zur formalen Spezifikation objektorientierter Systeme
  - Object-Z Klasse spezifiziert
    - die möglichen Zustände eines Klassenobjekts
    - die möglichen Zustandsübergänge
    - die Kommunikation des Objekts mit seiner Umgebung
    - Systeme durch Identifikation der zugehörigen Objekte und deren Interaktion
  - Wiederverwendung möglich durch
    - Vererbung
    - Instantiierung

Angewandte Telematik 8

## 5.1 Was ist Object-Z?

- Object-Z
  - Erweiterung der formalen Spezifikationsprache Z auf konservative Weise, d.h. Syntax und Semantik von Z bleiben in Object-Z erhalten
  - Z wiederum geht zurück auf die formale Spezifikation VDM (Vienna Development Method)
  - Entwickelt von einer Forschungsgruppe am Software Verification Research Centre, University of Queensland, Australien

## 5.2 Ein einführendes Beispiel

- Spezifikation eines Systems zur Verwaltung von Kreditkartenkonten
- D.h. Spezifikation der Basisfunktionalität der Kreditkartenkontenobjekte und deren Interaktion
  - Spezifikation der Kontenobjekte, deren Zustände und Operationen
    - Kontenobjekt gekennzeichnet durch zwei Zahlen:
      - Aktueller Kontostand
      - Kreditlimit (€1000, €2000, €5000)
    - Operationen
      - Geld abheben
      - Mit Karte einkaufen
      - Geld einzahlen
  - Spezifikation des Systems und Operationen zur Definition von Aktionen zwischen den Objekten

## 5.2.1 Das Object-Z Klassenkonstrukt

- Eine Klasse besteht aus 6 Teilen
  - Klassenname und -box
  - Sichtbarkeitsliste („Visibility List“)
  - Definition von Konstanten
  - Definition von Objektzuständen und Veränderungen („State Schema“)
  - Definition initialer Bedingungen („Initial Schema“)
  - Definition von Operationen auf Objekten („Operation Schemas“)

## 5.2.1 Das Object-Z Klassenkonstrukt

## 5.2.1 Das Object-Z Klassenkonstrukt

- Visibility List
  - Führt Eigenschaften auf, die sichtbar in der Umgebung eines Objekt der Klasse CreditCard sind
  - Spezifiziert Schnittstelle des Objekts
  - Auf Elemente der Schnittstelle kann von außen zugegriffen werden
  - Falls keine Visibility List existiert, sind alle Eigenschaften implizit sichtbar

↑ (limit,balance,INIT,withdraw,deposit,withdrawAvail)

## 5.2.1 Das Object-Z Klassenkonstrukt

- Constant Definition (Open Schema Box)
  - Ist axiomatische Definition
    - Deklarationsteil (oben)
      - Deklariert Konstantennamen und -typ
    - Prädikatenteil (unten)
      - Formeln der Prädikatenlogik 1. Ordnung
      - Definiert Constraints auf den Konstanten

limit:  $\mathbb{N}$

limit  $\in \{1000,2000,5000\}$

## 5.2.1 Das Object-Z Klassenkonstrukt

- State Schema
  - Deklaration von Zustandsvariablen (oben)
  - Prädikate auf Zustandsvariablen und Konstanten (unten)
  - Zustandsvariablen und Konstanten bilden zusammen die *Attribute* einer Klasse
  - Prädikate sind *Klasseninvariante*, d.h. müssen stets erfüllt sein
  - Schema bestimmt eine Menge von gültigen Instanzen der Variablen (hier: balance)

```
balance: Z
balance + limit ≥ 0
```

## 5.2.1 Das Object-Z Klassenkonstrukt

- Initial Schema
  - Trägt stets den Namen INIT
  - Spezifiziert Prädikate auf Attributen
  - Prädikat zusammen mit den Klasseninvarianten definiert *initiale Bedingungen*
  - Objekte sind in sog. *initialer Konfiguration*, falls die Werte der Attribute die initialen Bedingungen erfüllen
  - Initial Schema ist eine Bedingung, keine Operation

```
INIT
balance = 0
```

## 5.2.1 Das Object-Z Klassenkonstrukt

- Operation Schemas
  - Spezifizieren die Veränderungen eines Objekts der Klasse CreditCard
  - $\Delta$  (variable): Liste von (sog. primären) Variablen, die durch die Operation verändert werden
    - Konstanten können hier also nicht auftauchen!
    - Nicht aufgeführte Variablen können nicht geändert werden
  - Einführung einer Menge von *Kommunikationsvariablen* (z.B. amount?), die Informationen zwischen Objekt und seiner Umgebung weitergeben
    - ?: Inputvariable
    - !: Outputvariable

## 5.2.1 Das Object-Z Klassenkonstrukt

- Operation Schemas ff.
  - Spezifikation der Attributwerte vor und nach einer Operation (z.B. withdraw spezifiziert den Wert von balance)
  - variable': variable nach Anwendung einer Operation
  - Prädikat (z.B.  $\text{amount?} \leq \text{balance} + \text{limit}$ ) nicht erfüllbar, so ist Operation nicht anwendbar

withdraw \_\_\_\_\_

$\Delta$  (balance)

amount?: N

$\text{amount?} \leq \text{balance} + \text{limit}$

$\text{balance}' = \text{balance} - \text{amount?}$

deposit \_\_\_\_\_

$\Delta$  (balance)

amount?: N

$\text{balance}' = \text{balance} + \text{amount?}$

withdrawAvail \_\_\_\_\_

$\Delta$  (balance)

amount!: N

$\text{amount!} = \text{balance} + \text{limit}$

$\text{balance}' = - \text{limit}$

## 5.2.1 Das Object-Z Klassenkonstrukt

- Operation Schemas ff.
  - Auswirkung einer Operation kann festgestellt werden durch Verknüpfung des Schema Prädikats mit den Klasseninvarianten
  - In Klasseninvarianten zusätzlich Variablen' einsetzen

withdraw
balance, balance': Z
amount?: N
balance + limit $\geq$ 0
balance' + limit $\geq$ 0
amount? $\leq$ balance + limit
balance' = balance - amount?

## 5.2.2 Instantiierung und Interaktion von Objekten

- Instanziierung und Interaktion von Objekten
  - Bis jetzt: Ein Objekt
  - Jetzt: Menge von Objekten und deren Interaktion modellieren

## 5.2.2 Instantiierung und Interaktion von Objekten

TwoCards

↑ (totalbal,INIT,withdraw<sub>1</sub>,transfer,withdrawEither,replaceCard<sub>1</sub>,...)

$c_1, c_2$ : CreditCard

Δ

totalbal: Z

$c_1 \neq c_2$

totalbal =  $c_1$ .balance +  $c_2$ .balance

INIT

$c_1$ .INIT

$c_2$ .INIT

withdraw<sub>1</sub>  $\hat{=}$   $c_1$ .withdraw

transfer  $\hat{=}$   $c_1$ .withdraw  $\wedge$   $c_2$ .deposit

withdrawEither  $\hat{=}$   $c_1$ .withdraw []  $c_2$ .withdraw

replaceCard<sub>1</sub>

Δ ( $c_1$ )

card?: CreditCard

card?  $\notin$  { $c_1, c_2$ }

card?.limit =  $c_1$ .limit

card?.balance =  $c_1$ .balance

$c_1' = \text{card?}$

[weitere Operationen...]

## 5.2.2 Instantiierung und Interaktion von Objekten

- Objekte als Attribute einer Klasse
  - $c_1, c_2$  sind Attribute und zugleich Objekte der Klasse CreditCard, die Werte dieser Attribute sind Objektidentitäten
  - $c_1, c_2$  sind zwei unterschiedliche Objekte
  - Identität (nie änderbar) unterscheidet sich vom Zustand (änderbar) eines Objekts
- Sekundäre Variablen
  - Platziert **unter** dem Δ Symbol
  - Wert ändert sich entsprechend der Änderungen von Konstanten und primären Variablen
  - Verwendung von object.feature: feature muss in visibility list des Objekts enthalten sein

$c_1, c_2$ : CreditCard

Δ

totalbal: Z

$c_1 \neq c_2$

totalbal =  $c_1$ .balance +  $c_2$ .balance

## 5.2.2 Instantiierung und Interaktion von Objekten

- Initial Schema

- Objekte der Klasse TwoCards sind in initialer Konfiguration, wenn Kreditkartenobjekte in initialer Konfiguration sind

```
INIT  
c1.INIT  
c2.INIT
```

- Promote von Operationen

- $\text{withdraw}_1$  ist Operation von TwoCards  $\text{withdraw}_1 \triangleq c_1.\text{withdraw}$
- wird als Anwendung der withdraw-Operation auf  $c_1$  definiert (sog. Promote)

- Applikation von Operationen auf Objekte

- Ausdruck `object.operation`

## 5.2.2 Instantiierung und Interaktion von Objekten

- Verknüpfungsoperator  $\wedge$  (conjunction operator)

- Wert amount? ist Input für withdraw UND deposit, d.h. wird von c1 abgehoben und auf c2 eingezahlt

$$\text{transfer} \triangleq c_1.\text{withdraw} \wedge c_2.\text{deposit}$$

- Verknüpfungsoperator ist kommutativ und assoziativ
- Kommunikationsvariablen werden bei Verknüpfung gleichgesetzt

- Bsp.:  $\text{withdrawAvailBoth} \triangleq c_1.\text{withdrawAvail} \wedge c_2.\text{withdrawAvail}$

- Operation nur anwendbar, wenn gilt

$$c_1.\text{balance} + c_1.\text{limit} = c_2.\text{balance} + c_2.\text{limit}$$

- Durch Umbenennung von amount durch other umgehen

$$\text{withdrawAvailBoth} \triangleq c_1.\text{withdrawAvail} \wedge c_2.\text{withdrawAvail}[\text{other!}/\text{amount!}]$$

## 5.2.2 Instantiierung und Interaktion von Objekten

- Auswahloperator [] (choice operator)
  - Wählt zwischen zwei Operationen aus
$$\text{withdrawEither} \triangleq c_1.\text{withdraw} [] c_2.\text{withdraw}$$
  - Bedingungen:
    - Gleiche Kommunikationsvariable für beide Komponenten
    - Sind beide Operationen nicht anwendbar (weil bei beiden Konten der Betrag das Kontolimit übersteigt), ist die gesamte Operation nicht anwendbar
    - Ist genau eine der Operationen anwendbar (weil bei einem Konto der Betrag das Kontolimit nicht übersteigt), wählt der Auswahloperator diese Operation
    - Sind beide Operationen anwendbar, wählt der Auswahloperator eine dieser Operationen (sog. nichtdeterministische "angelic choice"), d.h. späte Auswertung, sodass möglichst weitgehende Auswertung ermöglicht wird
- Auswahloperator ist kommutativ und assoziativ

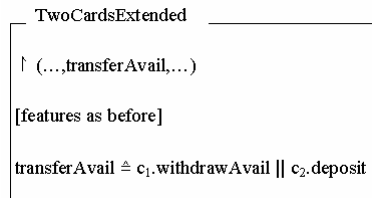
## 5.2.2 Instantiierung und Interaktion von Objekten

- Objektersetzung
  - $\text{replaceCard}_1$  modelliert Ersetzung von Objekt  $c_1$  durch Objekt  $\text{card}?$

$\text{replaceCard}_1$ _____
$\Delta(c_1)$
$\text{card}?: \text{CreditCard}$
$\text{card}? \notin \{c_1, c_2\}$
$\text{card}?.\text{limit} = c_1.\text{limit}$
$\text{card}?.\text{balance} = c_1.\text{balance}$
$c_1' = \text{card}?$

## 5.2.3 Inter-Objekt-Kommunikation

- Paralleloperator || (parallel operator)
  - Operator zur Modellierung der Interobjekt-Kommunikation



- transferAvail modelliert Abbuchung des max. Betrags von  $c_1$  und Buchung auf  $c_2$
- Komposition der beiden Operationen withdrawAvail und deposit durch Paralleloperator ||

## 5.2.3 Inter-Objekt-Kommunikation

- Parallel-Operator verhält sich wie Verknüpfungsoperator, aber erlaubt Inter-Objekt-Kommunikation zwischen den Inputs und Outputs der Operationen, in beide Richtungen. Namensgleichheit ist bedeutsam.
- Kommunikation über identische Kommunikationsvariablen
- Bei verschiedenen Kommunikationsvariablen entspricht Paralleloperator dem Verknüpfungsoperator
- Inter-Objekt-Kommunikation funktioniert in beide Richtungen (sogar in einer Operation)
- Paralleloperator ist kommutativ, aber nicht assoziativ

### 5.2.3 Inter-Objekt-Kommunikation

- Im Fall der Operation *transferAvail* wird zunächst *withdrawAvail* der Klasse *CreditCard* auf das Objekt *c1* angewendet.
- Ergebnis: *amount!*
- Parallel wird *deposit* auf *c2* angewendet. Das erfordert eine Eingabe *amount?*
- Da *amount!* und *amount?* gleichgesetzt werden, kommt es hier zu der versteckten Kommunikation im Rahmen des Parallel-Operators.
- Keine Assoziativität, da versteckte Kommunikation nur zwischen 2 Operationen möglich.
- Ohne Inter-Objekt-Kommunikation verhält sich der Parallel-Operator wie der Verknüpfungsoperator.

### 5.2.4 Objekt-Aggregation