Formale Modellierung
Vorlesung 12 vom 24.06.13: Formale Modellierung mit UML und OCL

Serge Autexier & Christoph Lüth

Universität Bremen

Sommersemester 2013

Fahrplan

- Teil I: Formale Logik
- Teil II: Spezifikation und Verifikation
  - Modellierung von Programmen
  - Die Z-Notation
  - Formale Modellierung mit der UML und OCL
- Teil III: Schluß

Das Tagesmenü

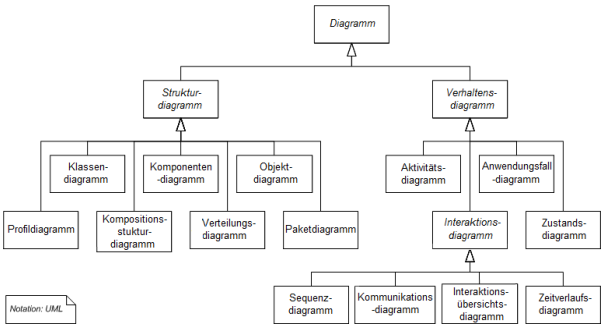
	Datenmodellierung	Programmbegriff
Isabelle/HOL	induktive Datentypen	totale rekursive Funktionen
Z	(induktive) Mengen	Vor/Nachzustand
UML	?	?

- Formale Modellierung mit der UML
- . . . mit der OCL: Object Constraint Language

UML als formale Spezifikationssprache

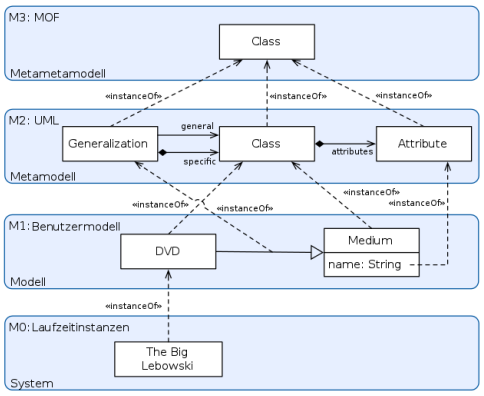
Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	Nein
Use-Case-Diagramm	Ablauf en gros	Nein
Aktivitätsdiagramm	Ablauf en detail	Nein
Zustandsdiagramm	Zustandsübergänge	Ja
Sequenzdiagramm	Kommunikation	Ja
Kommunikationsdiagramm	Struktur der Kommunikation	(Ja)
Zeitverlaufsdiagramm	Echtzeitaspekte	(Ja)

Diagramme in UML 2.3



Quelle: Wikipedia

Semantik der UML: Metamodellierung



Quelle: Wikipedia

OCL

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

OCL Basics

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

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`

Undefiniertheit in OCL

- Undefiniertheit propagiert (alle Operationen strikt) → OCL-Std. §7.5.11
- Ausnahmen:
  - Boolsche Operatoren (`and`, `or` beidseitig nicht-strikt)
  - Fallunterscheidung
  - Test auf Definiertheit: `oclIsUndefined` mit

$$\texttt{oclIsUndefined(e)} = \begin{cases} \texttt{true} & e = \bot \\ \texttt{false} & \text{otherwise} \end{cases}$$

- Resultierende Logik: dreiwertig

Dreiwertige Logik

- Wahrheitstabelle (starke Kleene-Logik, K_3):

	$\neg$
$\bot$	$\bot$
0	1
1	0

$\wedge$	$\bot$	0	1
$\bot$	$\bot$	0	$\bot$
0	0	0	0
1	$\bot$	0	1

$\vee$	$\bot$	0	1
$\bot$	$\bot$	$\bot$	1
0	$\bot$	0	1
1	1	1	1

$\longrightarrow$	$\bot$	0	1
$\bot$	$\bot$	$\bot$	1
0	1	1	1
1	$\bot$	0	1

$\longleftrightarrow$	$\bot$	0	1
$\bot$	$\bot$	$\bot$	$\bot$
0	$\bot$	1	0
1	$\bot$	0	1

- Fun Fact: K_3 hat keine Tautologien.
- Alternative: schwache Kleene-Logik (alle Operatoren strikt)

OCL Typen

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

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`

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`

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

Modelltypen

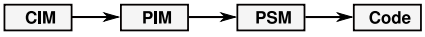
- Aus Attribute, Operationen, Assoziationen des Modells
- Navigation entlang der Assoziationen
- Für Kardinalität 1 Typ `T`, sonst `Set(T)`
- Benutzerdefinierte Operationen in Ausdrücken müssen zustandsfrei sein (Stereotyp `<<query>>`)

Style Guide

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

MDA + OCL

- MDA: Model-driven architecture
- Entwicklung durch Modelltransformation



- Rolle der OCL:
  - Metasprache
  - Codegenerierung
  - Laufzeitchecks
- Beispiele für Werkzeuge: MDT/OCL
  - MDT/OCL: EMF mit OCL-Unterstützung

Zusammenfassung

- Kritik UML:
  - “OO built-in”
  - Adäquat für eingebettete Systeme, CPS, . . . ?
- OCL erlaubt Einschränkungen auf Modellen
- Erlaubt mathematisch präzisere Modellierung
- Frage:
  - Werkzeugunterstützung?
  - Ziel: Beweise, Codegenerierung, . . . ?