Formale Modellierung
Vorlesung vom 16.04.12: Einführung

Till Mossakowski & Christoph Lüth

Universität Bremen

Sommersemester 2012

Organisatorisches

- Veranstalter:

Till Mosskawski	Christoph Lüth
till.mossakowski@dfki.de	christoph.lueth@dfki.de
Cartesium 2.51, Tel. 64216	MZH 3110, Tel. 59830

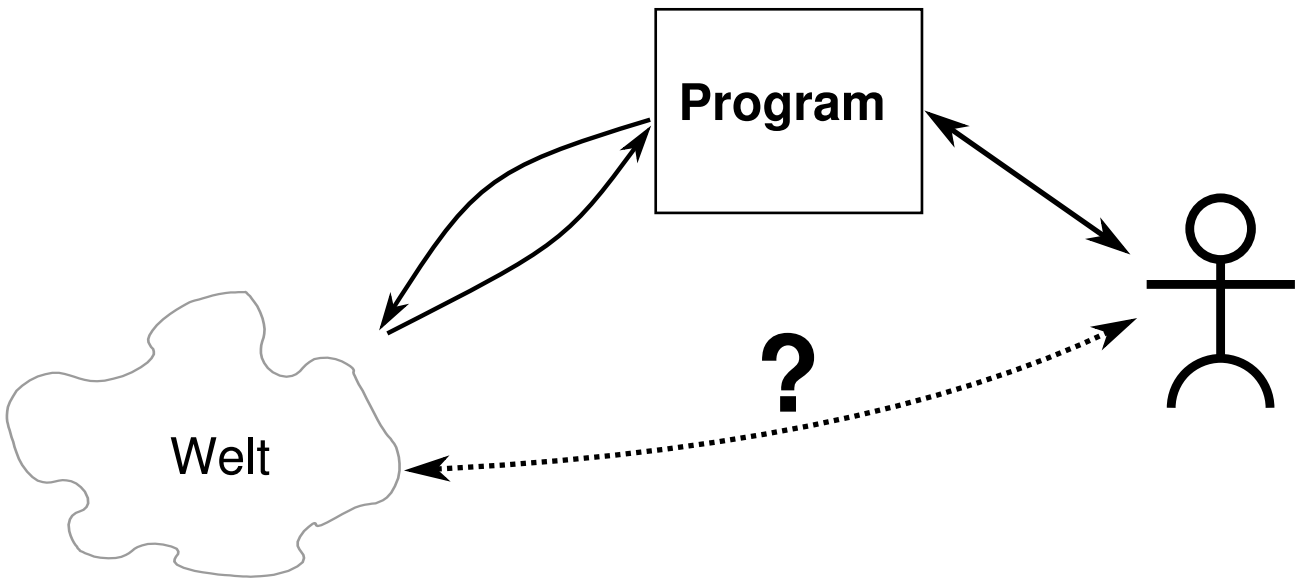
- Termine: Vorlesung: Montag, 14 – 16, Cartesium 2.43
  Übung: Donnerstag, 8 – 10, GW1 C1070

Therac-25

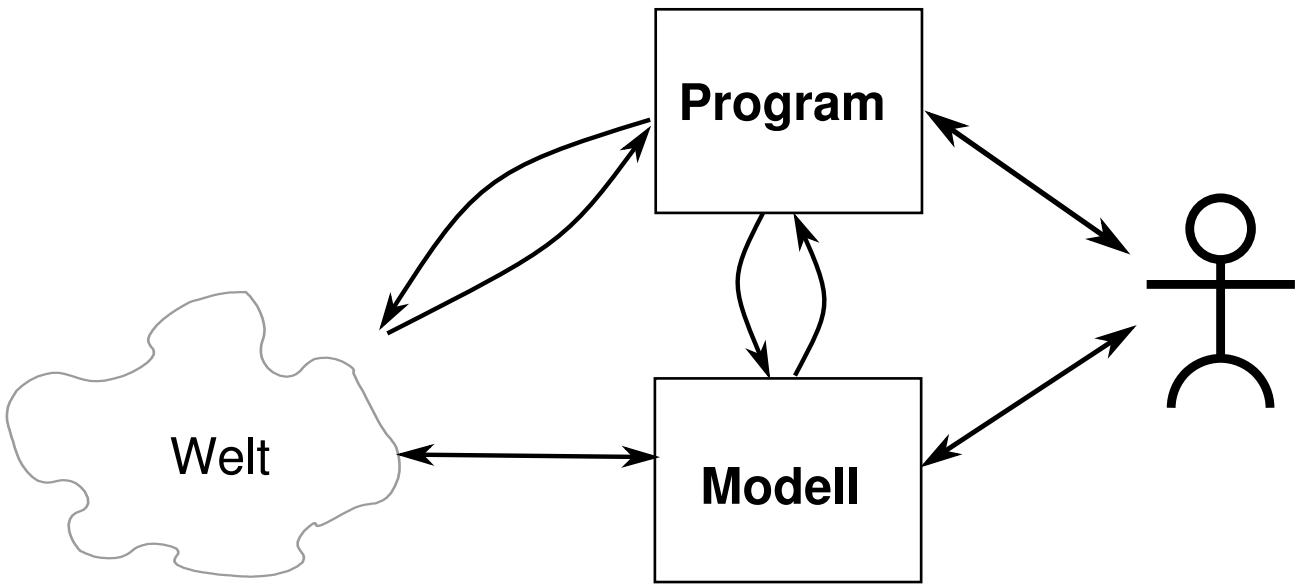
- Neuartiger Linearbeschleuniger in der Strahlentherapie.
  - Computergesteuert (PDP-11, Assembler)
- Fünf Unfälle mit Todesfolge (1985– 1987)
  - Zu hohe Strahlendosis (4000 – 20000 rad, letal 1000 rad)
- Problem: Softwarefehler
  - Ein einzelner Programmierer (fünf Jahre)
  - Alles in Assembler, kein Betriebssystem
  - Programmierer auch Tester (Qualitätskontrolle)

Ariane-5

Das Problem



Das Problem



Lernziele

1. Modellierung — Formulierung von Eigenschaften

Lernziele

1. Modellierung — Formulierung von Eigenschaften

2. Spezifikation — Eigenschaften von Programmen

Lernziele

1. Modellierung — Formulierung von Eigenschaften
2. Spezifikation — Eigenschaften von Programmen
3. Verifikation — Beweis der Eigenschaften

Lernziele

1. Modellierung — Formulierung von Eigenschaften
2. Spezifikation — Eigenschaften von Programmen
3. Verifikation — Beweis der Eigenschaften
4. Vertrautheit mit aktuellen Techniken: JML, UML, Temporallogik

Techniken der Modellierung und Spezifikation

- Annotationen an den Code
  - JML für Java: Abstraktion von konkreter Implementation
- Abstraktion vom Quellcode:
  - Abstrakte Programmmodelle: Zustandsautomaten, Klassendiagrame
- Abstraktion vom Ausführungsmodell:
  - Hier: Nebenläufigkeit (Temporallogik)

Java Modeling Language (JML)

- Zentral: funktionale Korrektheit
- Design by contract
- Spezifikation nahe am Code
- Hoare-Logik (pre/post conditions)
- Werkzeuge: ESC/Java2, Mobius

Unified Modeling Language (UML)

- allgemeine Modellierungssprache
- Spezifikation problemorientierter
- Übersetzung in verschiedene Programmiersprachen möglich
- Nur bestimme Aspekte sind formal
- hier: State Machines und Object Constraint Language (OCL)
- Werkzeuge: Hugo/RT und argouml

Temporallogik

- Spezifikation nebenläufiger Programme
- Sicherheits- und Fairness-Eigenschaften
- Werkzeug: nuSMV Modelchecker

Aufbau der Vorlesung

- Plan:
  - Woche 01 – 05 : Codeannotation mit JML
  - Woche 06 – 11 : Modellierung mit UML
  - Woche 11 – 14 : Modellchecking mit NuSMV
- Vorlesung und Übung dynamisch im Wechsel
- ca. 5 Übungsblätter, Gruppengröße max. 3
- Übungen werden vorgestellt und besprochen
- Scheinkriterien:
  - Übungsblätter bearbeiten und Fachgespräch
  - oder mündliche Prüfung

Formale Modellierung
Vorlesung vom 07.05.12: Beyond JML

Till Mossakowski & Christoph Lüth

Universität Bremen

Sommersemester 2012

Heute im Programm

- Grenzen der JML
- Nach JML: UML (und darüber hinaus)
- Zwei Fallbeispiele

Fallbeispiel 1: Bankautomat

Informelle Spezifikation

Nach dem Einlegen der Karte (cashcard o.ä.) kann der Benutzer auswählen zwischen dem Abheben von Bargeld, und der Anzeige des Kontostands.
In beiden Fällen liest der Automat von der Karte die Kontonummer des Benutzers. Beim Abheben gibt der Benutzer zuerst die gewünschte Summe und danach zur Authentifzierung sein PIN ein. Der Automat liest von der Karte die verschlüsselte PIN, und prüft ob die eingegebene mit der verschlüsselten übereinstimmt. Ist dieser der Fall, und ist das Konto gedeckt, wird das Bargeld zum Abheben bereitgestellt; danach wird die Karte zurückgegeben. Ist das nicht der Fall, wird eine entsprechende Fehlermeldung ausgegeben, und die Karte zurückgegeben. In beiden Fällen geht der Automat danach wieder in den Anfangszustand.

Kritik JML

- Implementationsnah
- Struktur der Spezifikation = Struktur der Implementierung
  - Hier: Events = Klassen?
- Mangelnde Abstraktion
- Nächster Schritt: UML

UML als formale Spezifikationssprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	

UML als formale Spezifikationssprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	

UML als formale Spezifikationssprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	

UML als formale Spezifikationssprache

Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	

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	

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	

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	

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	

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	

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	

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	

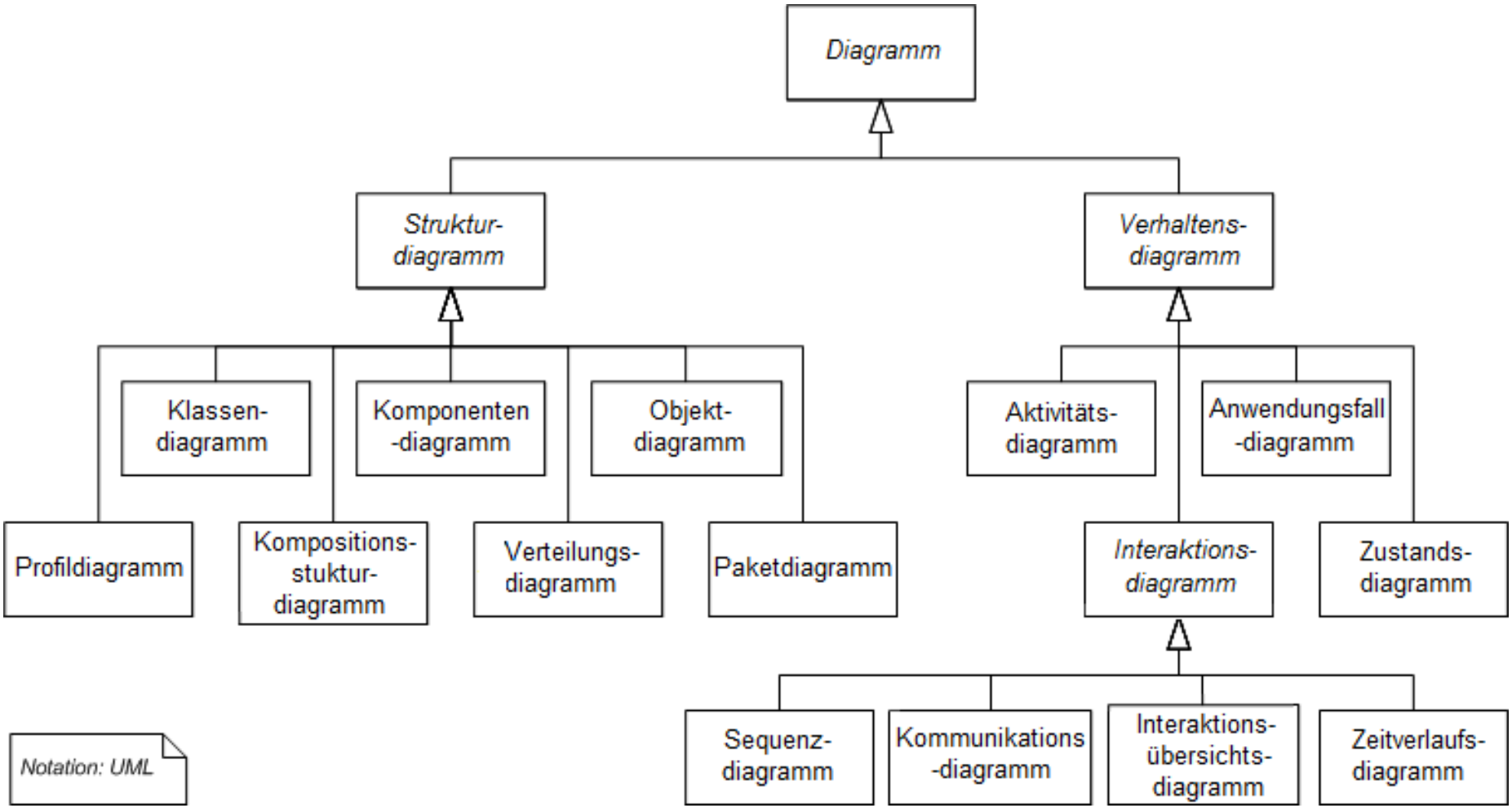
UML als formale Spezifikationssprache

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	

UML als formale Spezifikationssprache

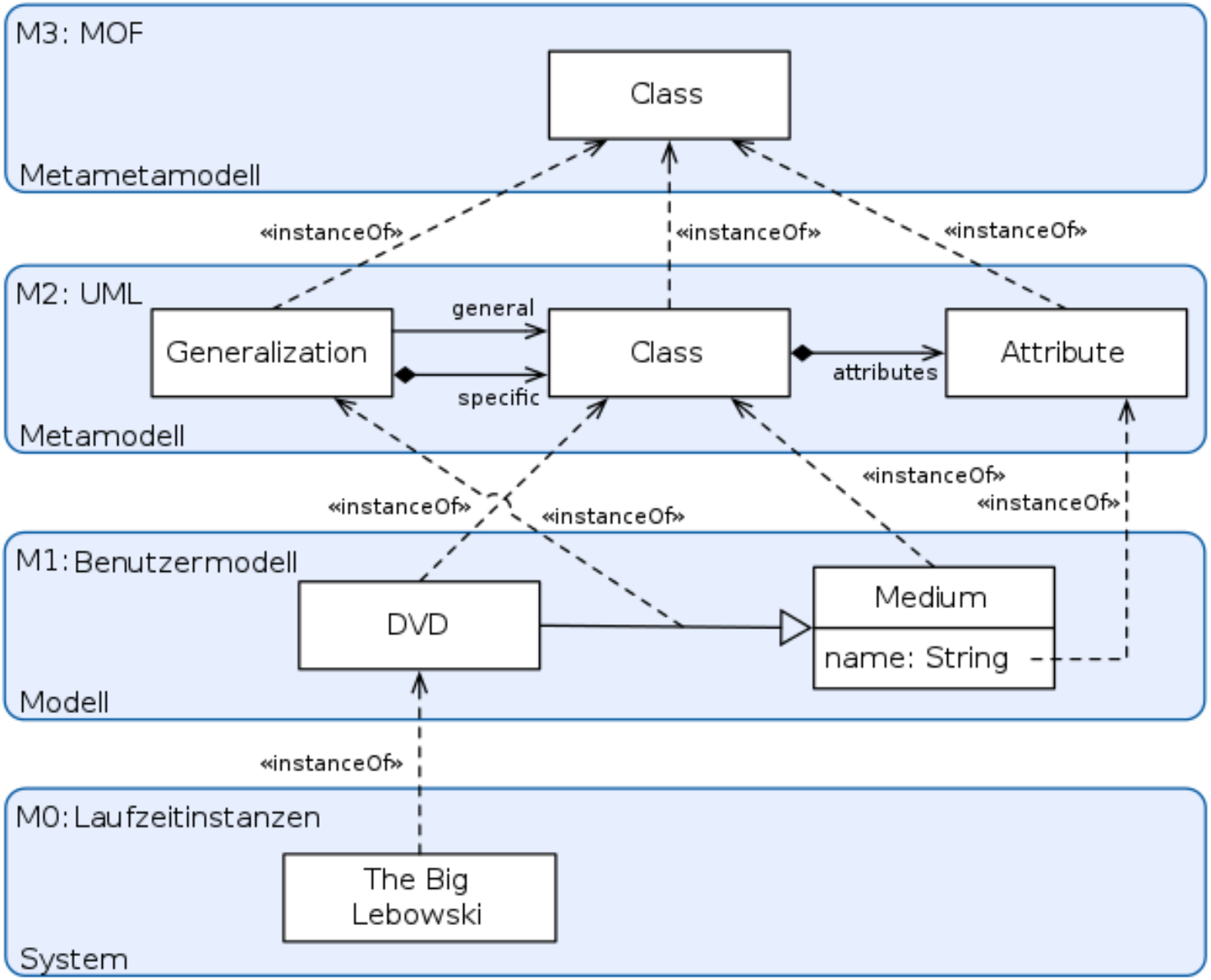
Diagrammtyp	Modellierte Aspekte	Formal
Klassendiagramm	Statische Systemstruktur	Ja
Paketdiagramm	Pakete, Namensräume	Nein
Objektdiagramm	Zustand von Objekten	(Ja)
Kompositionsstrukturdiagramm	Kollaborationen	Nein
Komponentendiagramm	Dynamische Systemstruktur	(Nein)
Verteilungsdiagramm	Implementierungsaspekte	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

Kritik UML

- “OO built-in”
- Adäquat für eingebettete Systeme, CPS, . . . ?

Fallbeispiel 2: omniRobot

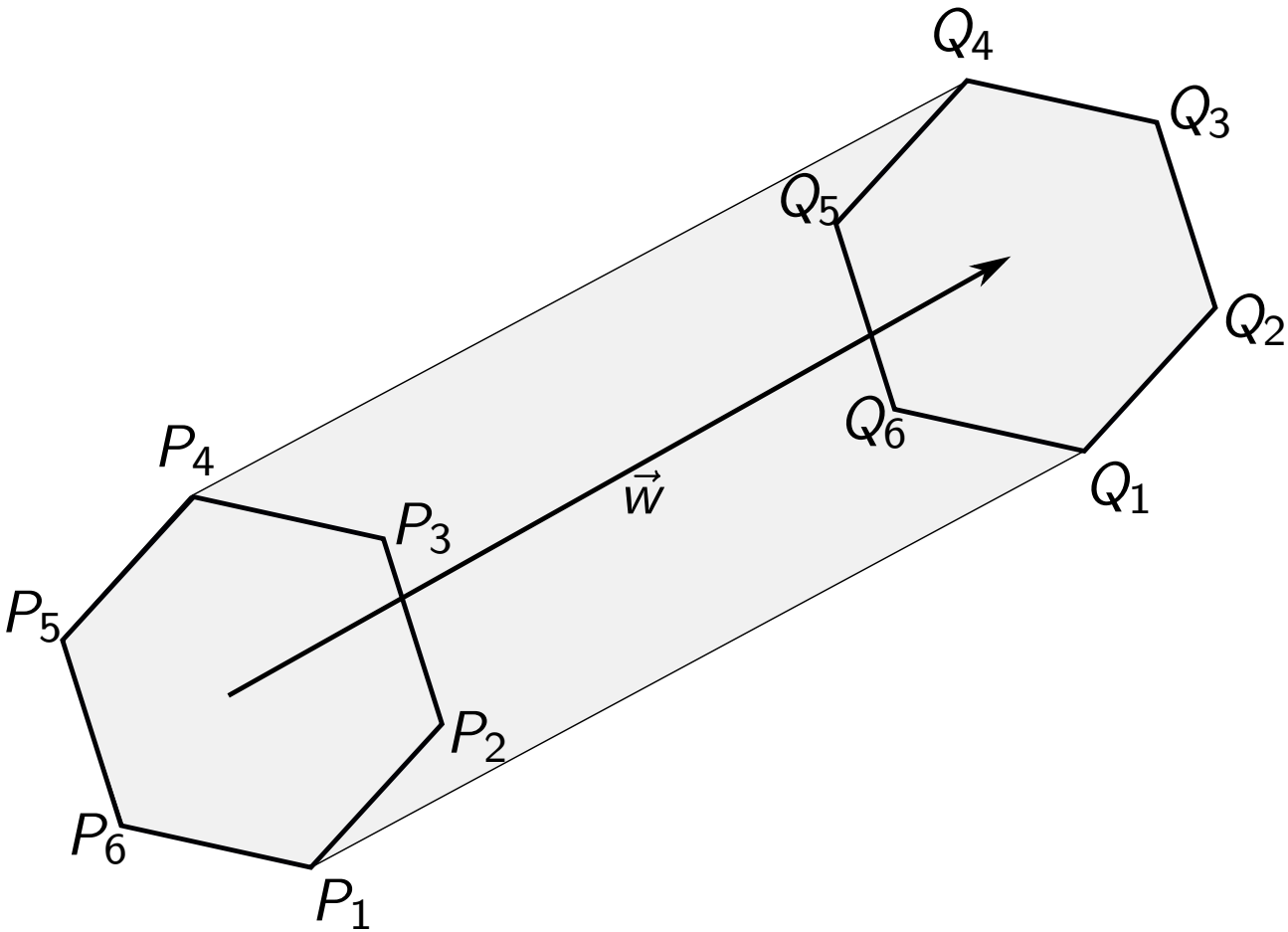
Informelle Spezifikation

Ein omnidirektionaler, autonomer Roboter soll gegen Kollisionen mit statischen Hindernissen gesichert werden.
Dazu wird zu jedem Zeitpunkt die mit der momentanen Geschwindigkeit $\vec{w}$ beim Bremsen bis zum Stillstand überstrichene Fläche A berechnet, die dann mit einer berührungslos wirkenden Schutzeinrichtung überwacht wird.

Quelle: Kuka AG

Modellierung

- Roboter als konvexes Polygon $\mathbf{K} = \langle \vec{K}_1, \ldots, \vec{K}_n \rangle$, momentane Pos. $\mathbf{P}$
- Momentane Geschwindigkeit: Vektor $\vec{v} = (v, \phi)$
- Bremsweg S_1, Anhalteweg S, als Vektor: $\vec{w} = (S, \phi)$
- Zu berechnen: Anhaltefläche A



Fazit Fallbeispiel 2

- Semantische Modellierung der realen Welt
- Nicht inhärent objektorientiert
- Modellierung in UML möglich
- Natürlicher: direkte Modellierung
- Fokus: Beweise, Manipulation von Formeln

Zusammenfassung

- JML sehr nahe an der Implementation
- UML abstrahiert von Implementation, aber nur bedingt von Anwendungsdomäne
- Nächste VL (Montag): Fallbeispiel 2 in UML und Z

Formale Modellierung
Vorlesung vom 14.05.12: Der sichere Roboter in Z

Till Mossakowski & Christoph Lüth

Universität Bremen

Sommersemester 2012

Heute im Programm

- Die Z Notation
- Modellierung des Sicheren Omnidirektionalen Roboters

Die Z Notation

- Basiert auf getypter Mengenlehre
- Entwickelt seit 1980 (Jean-Claude Abrial, Oxford PRG)
- Industriell genutzt (IBM, Altran Praxis (vorm. Praxis Critical Systems))
- LaTeX-Notation und Werkzeugunterstützung (Community Z Tools, HOL-Z)

Modell

- Bremszeit und Bremsstrecke (ab Bremszeitpunkt, $a_{brk} > 0$):

$$v(t) = v_0 - a_{brk}\,t \qquad s(t) = v_0\,t - \frac{a_{brk}}{2}\,t^2 \qquad \Longrightarrow \qquad T = \frac{v_0}{a_{brk}} \qquad S = \frac{v_0^2}{2a_{brk}}$$

- Modellierung in Z: Berechung des Bremsweges. Verzögerung t_L und Bremsverzögerung a_{brk} fest:

$$a_{brk}, t_L : \mathbb{Z}$$

Damit Brems- und Anhalteweg als Funktion über v:

$$brk : \mathbb{Z} \to \mathbb{Z}$$
$$stop : \mathbb{Z} \to \mathbb{Z}$$

$$\forall\, v : \mathbb{Z} \bullet brk\, v = v * v \operatorname{div} 2 * a_{brk}$$
$$\forall\, v : \mathbb{Z} \bullet stop\, v = v * t_L + brk(v)$$

Geometrie: Punkte

- Ein Punkt ist ein Schema mit Komponenten x und y:

$$\begin{array}{|l} \textit{POINT} \\ \hline x, y : \mathbb{Z} \\ \hline \end{array}$$

- Dazu einschlägige Operationen: Konstruktion aus Polarkoordinaten, Skalarmultiplikation:

$$\begin{array}{|l} \textit{polar} : \mathbb{Z} \times \mathbb{Z} \to \textit{POINT} \\ \hline \forall\, d, \omega : \mathbb{Z} \bullet (\textit{polar}(d, \omega)).x = d * \textit{cos}\, \omega \\ \forall\, d, \omega : \mathbb{Z} \bullet (\textit{polar}(d, \omega)).y = d * \textit{sin}\, \omega \\ \hline \end{array}$$

$$\begin{array}{|l} \textit{smult} : \mathbb{Z} \times \textit{POINT} \to \textit{POINT} \\ \hline \forall\, d : \mathbb{Z};\ p : \textit{POINT} \bullet (\textit{smult}(d, p)).x = d * p.x \\ \forall\, d : \mathbb{Z};\ p : \textit{POINT} \bullet (\textit{smult}(d, p)).y = d * p.y \\ \hline \end{array}$$

Mehr Standardoperationen auf Punkten

- Addition und Subtraktion von Punkten:

$$
\begin{array}{|l}
add : POINT \times POINT \to POINT \\
minus : POINT \times POINT \to POINT \\
\hline
\forall\, p, q : POINT \bullet (add(p, q)).x = p.x + q.x \\
\forall\, p, q : POINT \bullet (add(p, q)).y = p.y + q.y \\
\forall\, p, q : POINT \bullet minus(p, q) = add(p, smult(0 - 1, q))
\end{array}
$$

- Ganzzahlige Wurzel:

$$
\begin{array}{|l}
sqrt : \mathbb{Z} \to \mathbb{Z} \\
\hline
\forall\, i : \mathbb{Z} \bullet sqrt\, i * sqrt\, i \leq i \wedge i < (sqrt\, i + 1) * (sqrt\, i + 1)
\end{array}
$$

- Damit Betrag eines Punktes (als Vektor):

$$
\begin{array}{|l}
len : POINT \to \mathbb{Z} \\
\hline
\forall\, p : POINT \bullet len\, p = sqrt(p.x * p.x + p.y * p.y)
\end{array}
$$

Geometrie: Bewegung

- Bewegung: kombinierte Translation und Rotation

$$
\begin{array}{|l}
\mathit{movep} : \mathit{POINT} \times \mathbb{Z} \times \mathbb{Z} \to \mathit{POINT} \\
\hline
\forall\, p : \mathit{POINT};\ d, \omega : \mathbb{Z} \bullet \\
\qquad \mathit{movep}(p, d, \omega) = \mathit{add}(p, \mathit{polar}(d, \omega))
\end{array}
$$

- Bewegung eines Polygons:

$$
\begin{array}{|l}
\mathit{move} : \mathrm{seq}\, \mathit{POINT} \times \mathbb{Z} \times \mathbb{Z} \to \mathrm{seq}\, \mathit{POINT} \\
\hline
\forall\, p : \mathrm{seq}\, \mathit{POINT};\ d, \omega : \mathbb{Z} \bullet \\
\qquad \mathit{move}(p, d, \omega) = (\lambda\, i : \mathrm{dom}\, p \bullet \mathit{movep}(p(i), d, \omega))
\end{array}
$$

Geometrie: Konvexe Hülle

- Strecke zwischen zwei Punkten:

$$seg : POINT \times POINT \to \mathbb{P}\, POINT$$

$$\forall\, p, q : POINT \bullet seg(p, q) = \{r : POINT \mid \exists\, z : \mathbb{N} \bullet z \leq len(minus(p, q)) \wedge r = add(smult(z, p), smult(len(minus(p, q)) - z, q))\}$$

- Konvexität (Menge aller konvexen Punktmengen):

$$conv : \mathbb{P}(\mathbb{P}\, POINT)$$

$$conv = \{c : \mathbb{P}\, POINT \mid \forall\, p, q : c;\ r : POINT \bullet r \in seg(p, q) \Rightarrow r \in c\}$$

- Konvexe Hülle: kleinste p umfassende konvexe Menge

$$convhull : \mathbb{P}\, POINT \to \mathbb{P}\, POINT$$

$$\forall\, p : \mathbb{P}\, POINT \bullet convhull\, p = \bigcap \{q : \mathbb{P}\, POINT \mid q \in conv \wedge p \subseteq q\}$$

Der Roboter und seine Bremsfläche

- Der Roboter hat eine Kontour, Geschwindigkeit, Orientierung:

Robot

$$P : \operatorname{seq} POINT$$
$$v : \mathbb{Z}$$
$$\omega : \mathbb{Z}$$

$$\#P > 2$$
$$\operatorname{ran} P \in conv$$

- Die beim Anhalten überstrichene Fläche:

$$area : \operatorname{seq} POINT \times \mathbb{Z} \times \mathbb{Z} \to \mathbb{P}\, POINT$$

$$\forall\, v, \omega : \mathbb{Z};\ P : \operatorname{seq} POINT \bullet$$
$$area(P, v, \omega) = convhull\,(\operatorname{ran} P \cup \operatorname{ran}\,(move(P, stop\, v, \omega)))$$

Der sichere Roboter

- Menge von Hindernissen

$$obs : \mathbb{P}\, POINT$$

- Der sichere Roboter

SafeRobot

$$Robot$$

$$area(P, v, \omega) \cap obs = \emptyset$$

- Sicherheitseigenschaft:

SafeMove

$$\Delta Robot$$

$$SafeRobot \Rightarrow SafeRobot'$$

Bewegung des Roboters

- Geschwindigkeit bleibt gleich:

RobotSameSpeed

$\Delta Robot$

$v' = v$

$\omega' = \omega$

$P' = move(P, v, \omega)$

- Sicher?

SafeSameSpeed

RobotSameSpeed

$SafeRobot \Rightarrow SafeRobot'$

Bewegung des Roboters

- Geschwindigkeit erhöht sich, Lenkwinkel bleibt nur in Bewegung:

$$\delta v : \mathbb{Z}$$

RobotSpeedUp

$$\Delta Robot$$

$$v' \leq v + \delta v$$
$$v \neq 0 \Rightarrow \omega' = \omega$$
$$P' = move(P, v, \omega)$$

- Sicher?

SafeSpeedUp

$$RobotSpeedUp$$

$$SafeRobot \Rightarrow SafeRobot'$$

Zusammenfassung

- Z ist leichtgewichtige Notation
- Werkzeugunterstützung: LaTeX, Typcheck
- Nächste Woche: UML

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

Till Mossakowski & Christoph Lüth

Universität Bremen

Sommersemester 2012

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 (Kleene-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`

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>>`)

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 (Kleene-Logik)
- Iteratoren: “semi-strikt”

Style Guide

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

Zusammenfassung

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

Formale Modellierung
Vorlesung vom 04.06.12: Werkzeugunterstützung für OCL

Till Mossakowski & Christoph Lüth

Universität Bremen

Sommersemester 2012

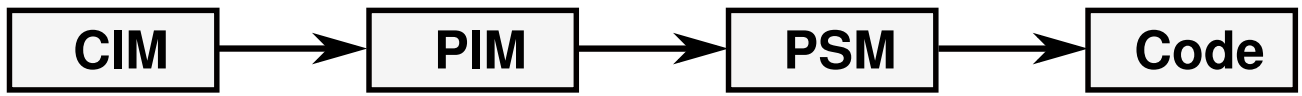
Werkzeuge für OCL

Klassifikation:

- MDA + OCL
- Specifikation mit OCL
- Interaktive Entwicklung mit OCL

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

Spezifikation

- USE (UML Specification Environment, Uni Bremen)
  - Spezifisch UML+OCL
  - Syntax/Typecheck
  - Animation
  - Konsistenzprüfung
- ArgoUML (Poseidon)
  - UML 1.5, alle Diagramme
  - OCL: Syntax/Typecheck

HOL-OCL

- Entwickelt an der ETH Zürich
- Flache Einbettung von OCL nach HOL
- Prüft Konsistenz der Modelle
- Präziser als OCL-Standard (z.B. Semantik von Mengen)
- Aufbauende Werkzeuge: Modelltransformationen

Zusammenfassung OCL-Werkzeuge

- OCL wird stiefmütterlich behandelt
- Generierung von Laufzeitprüfungen
- OCL als Abfragesprache
- Modelltransformation (used?)