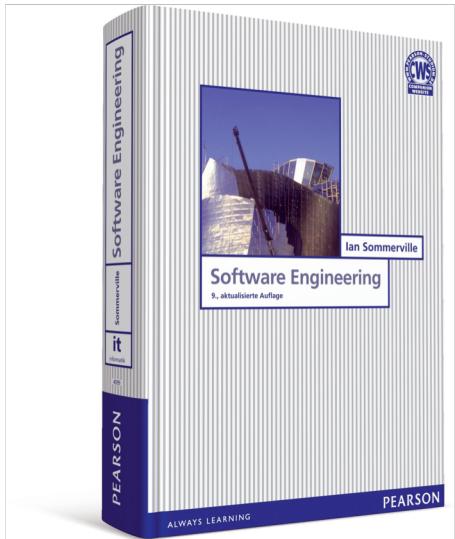


Teil 2: Meta-Modellierung und Domänenspezifische Sprachen

Literatur



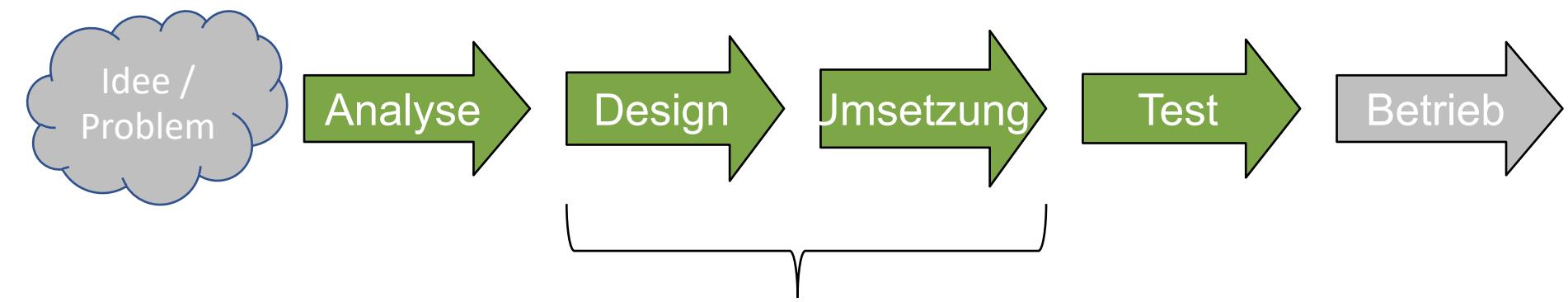
- Ian Sommerville, Software Engineering, 9. überarbeitete Ausgabe, Pearson 2012.
- Stahl und Völter, Modellgetriebene Softwareentwicklung, dpunkt.verlag GmbH, 2005.
- The Unified Modeling Language; Booch, Rumbaugh, Jacobsen; Addison-Wesley; 2001



Agenda

- Einleitung und Motivation
- Meta-Modellierung
- Domänen-spezifische Sprachen

Einordnung



**Meta-Modellierung und
Domänenspezifische Sprachen**

Wiederholung: Geschichte

- Die Idee einer Disziplin 'Software Engineering' entsteht bei einer NATO Konferenz 1968 / 1969 in einer Diskussion der Software Krise

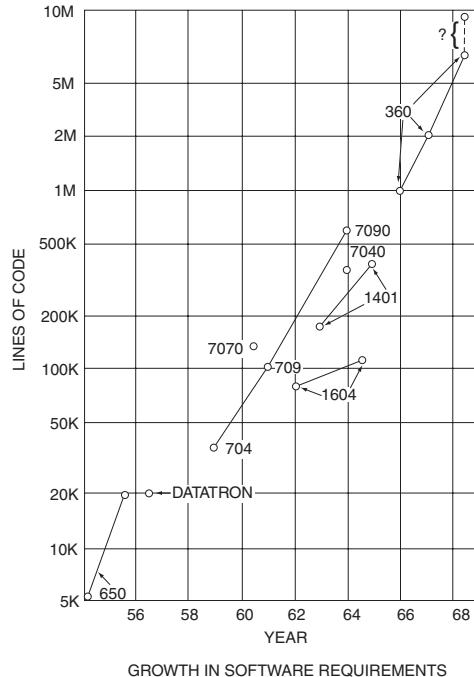
»Production of large software has become a scare item for management. By reputation it is often an unprofitable morass, costly and unending.

...

No less a person than T. J. Watson said that OS/360 cost IBM over 50 million dollars a year during its preparation, and at least 5000 man-years' investment.

...

The commitment to many software projects has been withdrawn. This is indeed a frightening picture.«



<http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF>

Fortsetzung: Projektkomplexität

Million Lines of Code

iOS app - simple game	0,01
Unix v 1.0 (1971)	0,01
Win32/Simile virus	0,01
iOS app - photo editing	0,04
Pacemaker	0,08
Photoshop v1 (1990)	0,12
Camino	0,20
Quake 3 engine	0,31
Space Shuttle	0,40
18000 pages of printed text	1,00
Crysis	1,00
War And Peace x 14, Ulysses x 25, The Catcher in The Rye x 63	1,00
Syphilis	1,14
Age of Empires Online	1,20
CESM Community Earth System Model	1,20
F-22 Raptor	1,70
Linux Kernel 2.2.0 (1999)	1,80
Hubble Space Telescope	2,00
Unreal Engine 3	2,00
Lines of code de-bugged in the Jurassic Park network by Dennis Nedry	2,00
Windows 3.1 (1992)	2,50
Drones (control software)	3,50
ROOT software (at the LHC)	3,50
Photoshop CS 6 (2012)	4,50
Windows NT 3.1 (1993)	4,50
HD DVD Players on XBox	4,70
HealthCare.gov - needed to repair	5,00
Mars Curiosity Rover	5,00
Linux kernel 2.6.0 (2003)	5,20
Google Chrome (estimate 2) (2011)	5,40

Quelle: <http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>

Fortsetzung: Projektkomplexität

	Million Lines of Code
Chevy Volt (electric car)	10,00
Intuit Quickbooks	10,00
Windows NT 4.0 (1996)	11,50
Android	12,00
Mozilla Core	12,50
MySQL	12,50
Boeing 787, total flight software	14,00
Android (upper estimate)	15,00
Linux 3.1 (recent version, 2013)	15,00
Apache Open Office	23,00
F-35 Fighter	24
Microsoft Office (2001)	25,00
Windows 2000 (2000)	29,00
Microsoft Office for Mac (2006)	30,00
Symbian	37,60
Windows 7	40,00
Windows XP (2001)	40,00
Microsoft Office (2013)	45,00
Large Hadron Collider	50,00
Microsoft Visual Studio 2012	50,00
Windows Vista (2007)	50,00
Facebook (without backend code)	62,00
US Army's Future Combat System	63,80
Debian 5.0 codebase	68,00
Mac OS X 10.4	86,00
Software in typical new car, 2013	100,00
Debian 5.0 (all software in package)	324,00
<u>Healthcare.gov</u>	500,00
Google	2.000,00

Quelle: <http://www.informationisbeautiful.net/visualizations/million-lines-of-code/>

Komplexität beherrschen

Ansätze, um **Komplexität zu beherrschen**:

Abstraktion:

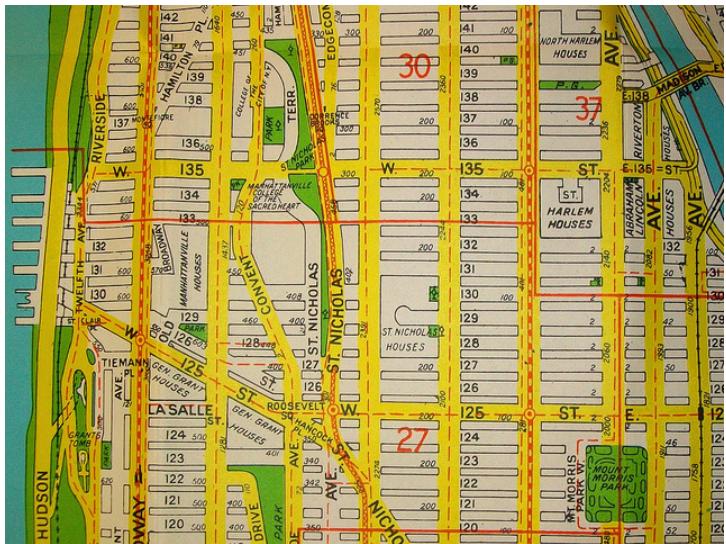
- Ausblenden von Detailinformation.
- Einsatz von geeigneten Modellen

Strukturierung / Modularisierung:

- Aufteilung in klar abgegrenzte Unterstrukturen.
- Partitionierung der Aufgaben (Dekomposition).

Abstraktion

Abstrakt:



Konkret:



<https://www.flickr.com/photos/eirikref/2302281782/>

<https://www.flickr.com/photos/23465812@N00/7968397600/>

<https://creativecommons.org/licenses/by/2.0/>

Modularisierung



<https://www.flickr.com/photos/martindew/7945608932/>
<https://creativecommons.org/licenses/by/2.0/>

Wdh.: Geschichte der Softwareentwicklung

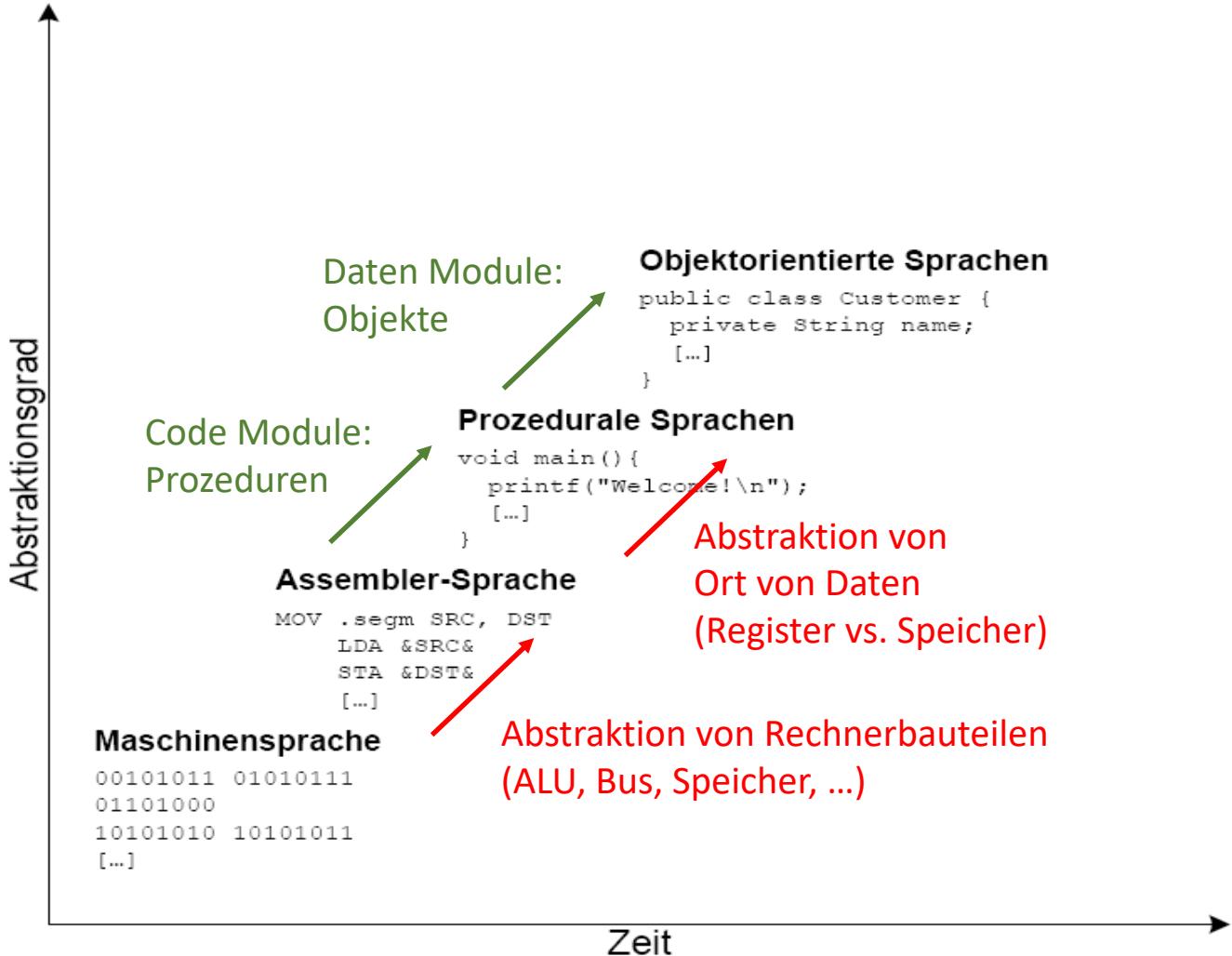
- Problem:
Komplexitäts-
Beherrschung

- Lösung:

Abstraktion

und

Modularisierung



Von Anforderungen zu Programmen

Anforderungen

Der Dienst erlaubt die Anmeldung

Das System ist Web-basiert

semantische Lücke

int x = 0;
Objekt println()



Objekt-orientierte Programme

Konsequenzen der Lücke: Fachlichkeit vs. Technik

- **Dominierung von Fachlichkeit durch Technik:**
 - Umsetzung fachlicher Basiskonzepte dominiert durch Technik.
 - Anwendungsentwickler:
 - Benötigen **umfangreiches technisches Wissen...**
 - ...statt sich auf fachliche Anwendungsdomäne zu konzentrieren.
 - Fachabteilung und Entwickler:
 - **Verständigung** auf völlig unterschiedlichen Abstraktionsebenen.
 - Abstraktionsniveau derzeitiger Entwicklungsansätze zu niedrig.

Konsequenzen der Lücke: Fehlende Durchgängigkeit

- **Methodischer Bruch** zwischen Analyse, Design und Implementierung:
 - Dokumentation nach Beginn Implementierung oft **nicht aktualisiert**.
 - Macht Dokumentation nahezu nutzlos.
 - **Erschwert Einarbeitung** neuer Mitarbeiter in späten Phasen.
 - **Erhöht Rüstzeiten** in Betriebs- bzw. Wartungsphase um Vielfaches.
- **Fehlende Nachverfolgbarkeit (Traceability)**:
 - Fehlende Durchgängigkeit für verschiedene Artefakte (Anforderungen, Dokumentation, Code) im Lebenszyklus.
 - Besonders in Bezug auf Anforderungen, fehlende
 - **Nachvollziehbarkeit** und
 - **Rückverfolgbarkeit**

Konsequenzen der Lücke: Schlussfolgerung

- Bedarf nach **höherem Abstraktionsniveau**.
- Forderung nach **durchgängigerer Auswahl der Ausdrucksmittel**.

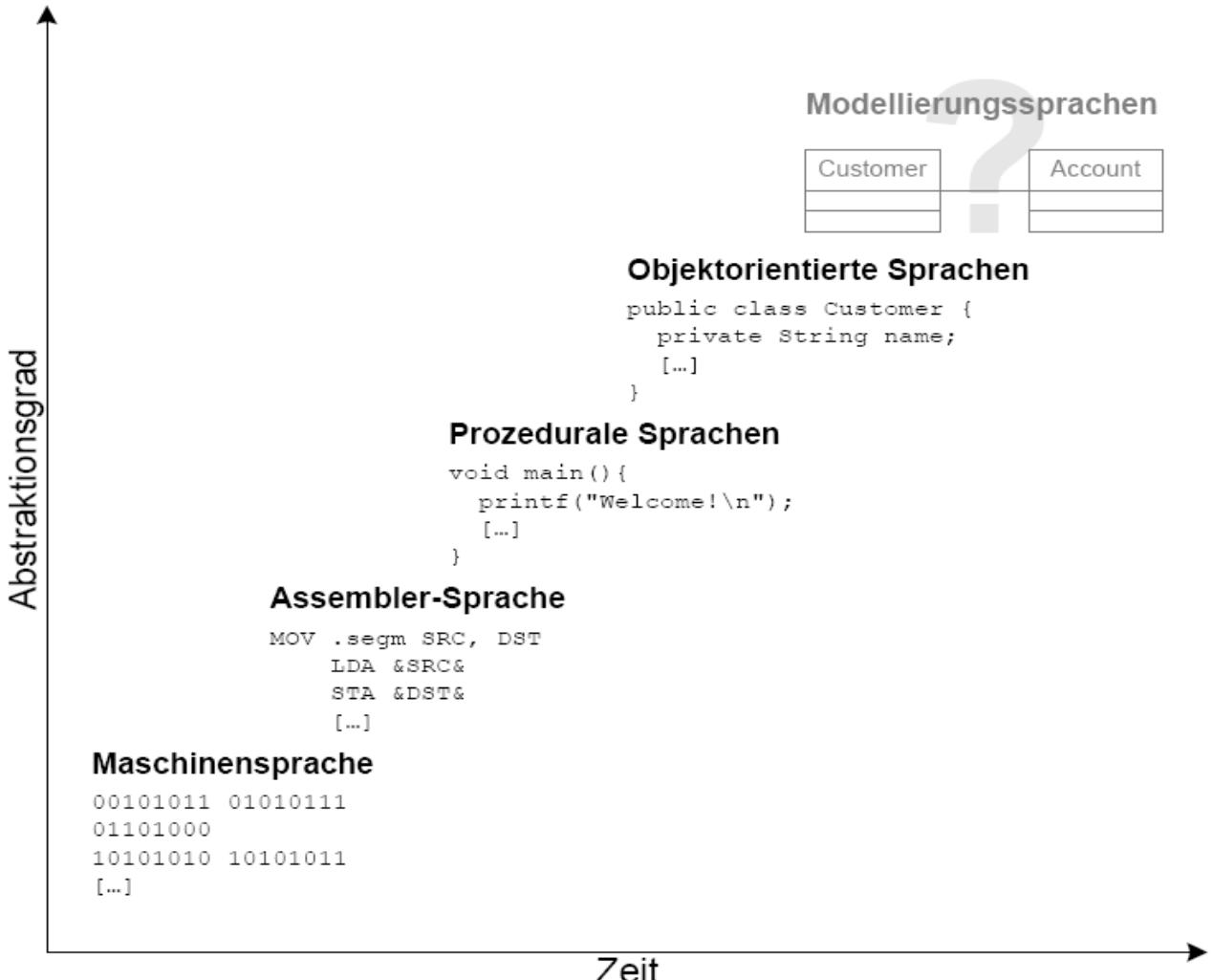
⇒ **Lösung: Modelle**

⇒ **Abstrahieren und fokussieren auf das Wesentliche.**

⇒ Brücke von **fachlicher Problemwelt** in **technische Lösungswelt**.

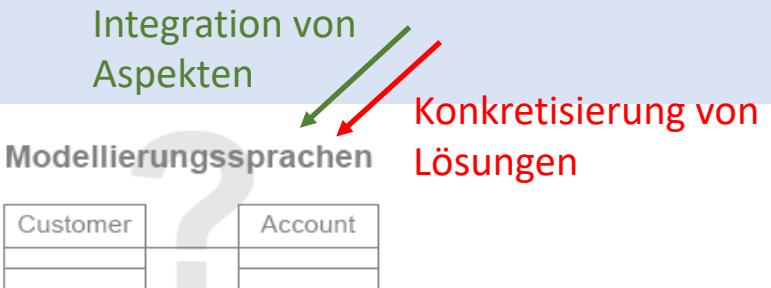
Modellierungssprachen

- Abstraktion:**
Weglassen von Details in Modellen
- Strukturierung:**
Modelle für Teilespektren oder Perspektiven

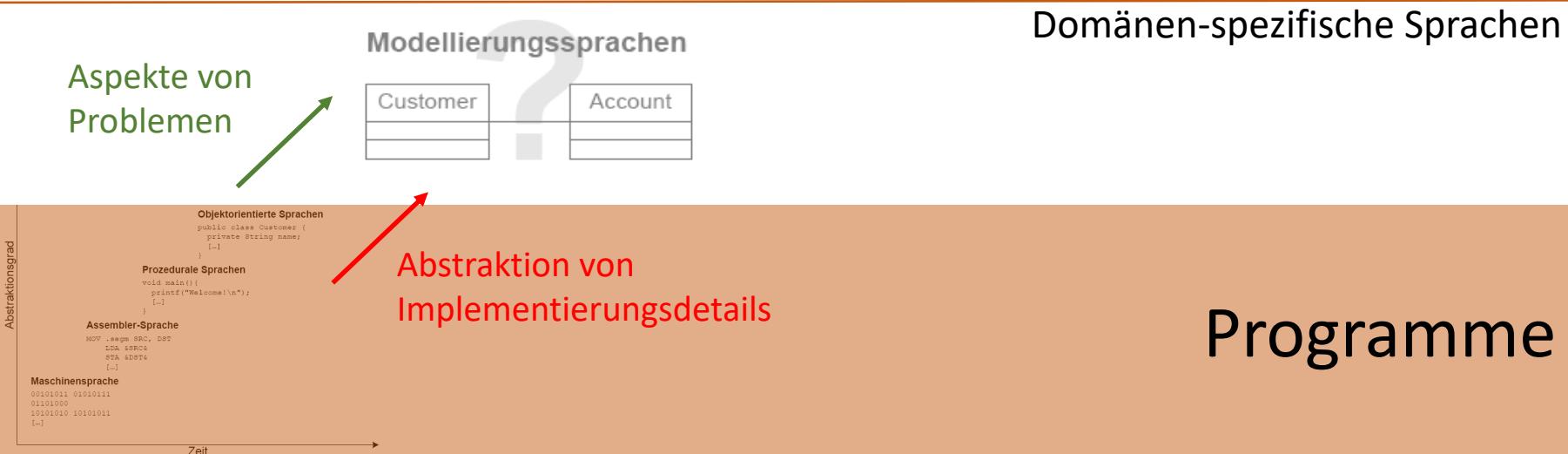


Von Anforderungen zu Programmen

Anforderungen



Modellgetriebene Software-Entwicklung



Model Driven Architecture

- Initiative der Object Management Group (OMG), 2001
- Konkreter Ansatz für modellbasierte Entwicklung.

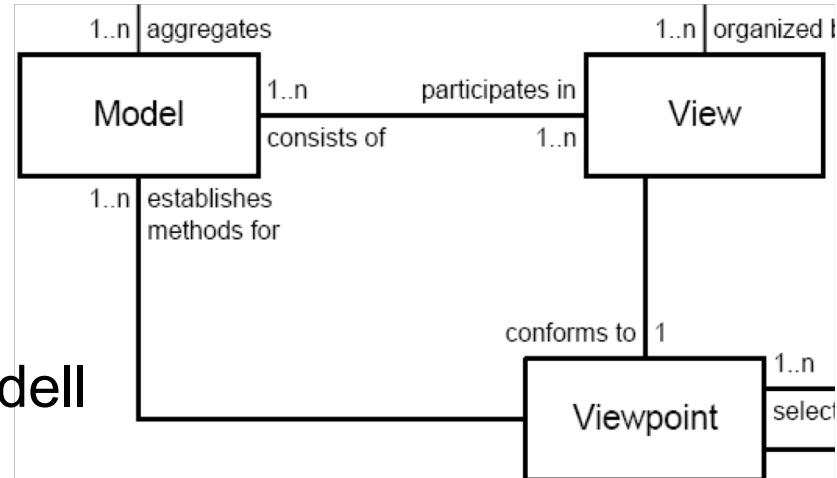
Kern Ideen:

- Spezifikation der Software unabhängig von technischer Umsetzung.
- Verschiedene Abstraktionslevel für Modelle:
 - Plattform-unabhängig (PIM).
 - Plattform-spezifisch (PSM).
- Übergang von abstrakten zu technologiespezifischen Modellen:
 - Vollautomatisiert: Verwendung von Transformationswerkzeugen.
 - Beschreibung der Transformationen in Transformationsbeschreibungen.

Unified Modeling Language

Unified Modeling Language (UML):

- **Modellierung, Dokumentation, Spezifizierung und Visualisierung von komplexen Softwaresystemen.**
- **Standard** der Object Management Group (OMG).
- **Unterschiedliche Modellierungskonzepte** auf einheitlicher Basis:
 - 14 Diagrammtypen in Version 2.4.1
- **Modell vs. Diagramm:**
 - UML-Modell besteht aus einem oder mehreren Diagrammen.
 - Ein Diagramm entspricht einer bestimmten Sicht („View“) auf Modell (vgl. IEEE 1471).



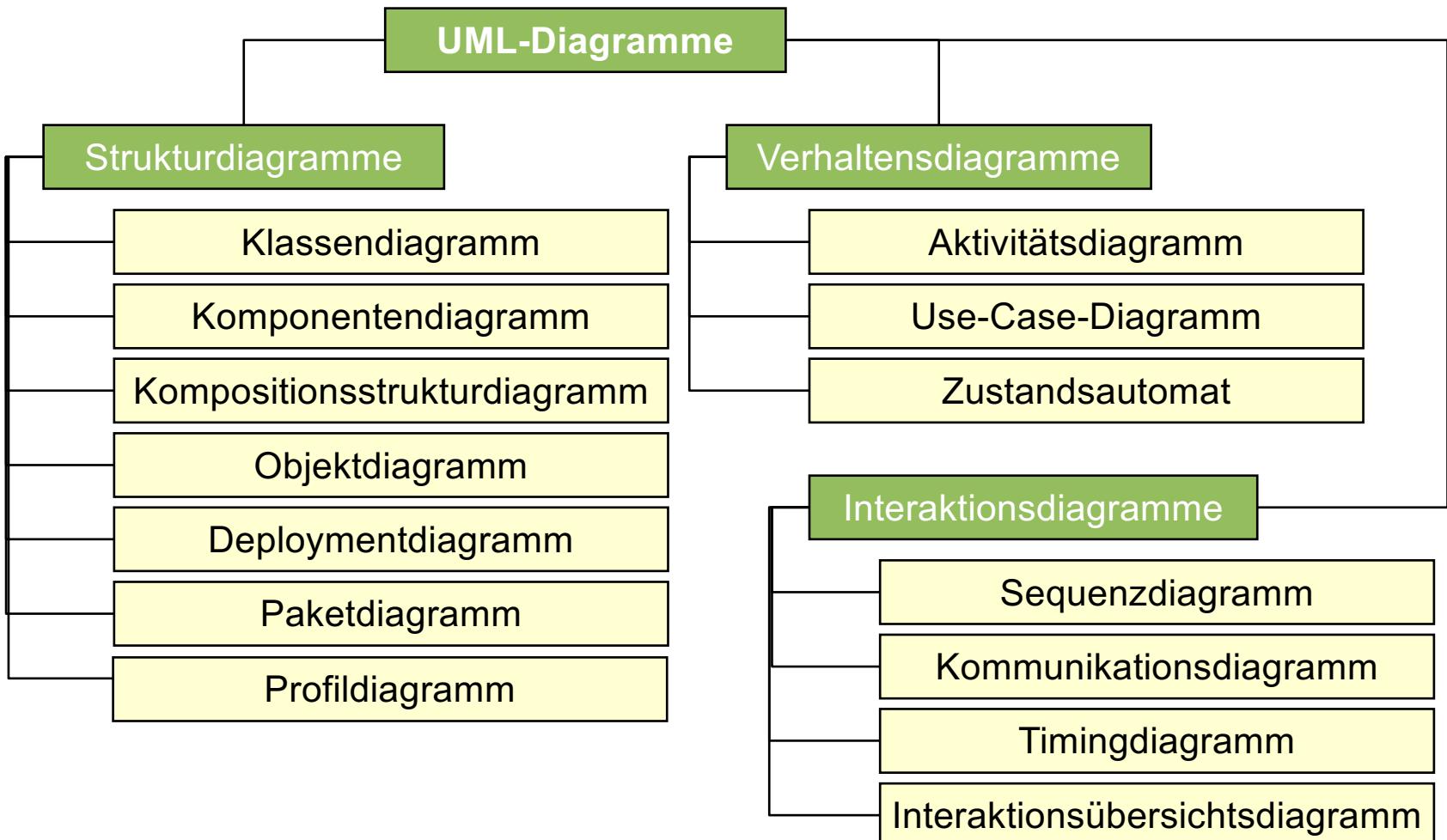
UML: Was Sie von SWT wiederholen sollten

- Was Sie sich für dieses Kapitel noch einmal anschauen sollten, um dahinterliegende **Metamodellierung** zu verstehen:
 - **UML-Klassendiagramme** (Generalisierung, Assoziationen, Komposition, Aggregation, Multiplizität,...)
 - **UML-Aktivitätsdiagramme** (Aktivitäten, Aktionen, Verzweigungsknoten, Verbindungsknoten, Verteilungsknoten, Kontrollflüsse,...)

Was auch hilfreich ist zu kennen:

- Weitere **Struktur- und Verhaltensdiagramme** der UML (z.B. Objekt-, Sequenz-, Zustandsdiagramme)

Wiederholung: Arten von UML-Diagrammen



Beispiele aus der Vorlesung

- Anforderungen:
 - Übergang Idee => Erwartung
 - Modelle: Use-Case Diagramme, Satz-Schablonen, ...
- Architektur / Dekomposition:
 - Strukturierung des Systems (Modularisierung und Abstraktion)
 - Modelle: Paket-Diagramme

Herausforderungen / Fragen

- Welche Elemente darf ein Modell enthalten
- Ist ein Modell korrekt?

- Was ist die Bedeutung eines Modells?
- Kann man aus Modellen Code generieren?

Agenda

- Einleitung und Motivation
- Meta-Modellierung
- Domänen-spezifische Sprachen

Meta?

- Wikipedia:
Meta- (from Greek: μετά = "after", "beyond", "with", "adjacent", "self")
- [Kühne]: ““meta” is used whenever an operation is applied twice”, e.g.
 - meta-discussion: a discussion about how to conduct a discussion
 - meta-learning: learning general learning strategies while learning a particular subject
 - meta-mathematics: applying mathematical methods to mathematics

Kühne, Matters of (Meta-) Modeling, STTT, Springer, 2006.

Stachowiak's General Modeling Theory

A model has three features [Stachowiak]:

1. **mapping feature** (“Abbildungsmerkmal”):

A model is always a model *of something*, that is, a mapping or representation of a natural or artificial original, which may in turn be a model itself.

2. **reduction feature** (“Verkürzungsmerkmal”):

A model usually does not capture *all* attributes of the original it represents, but only those which are relevant to its creator / user.

3. **pragmatic feature** (“Pragmatisches Merkmal”):

A model needs to be usable in place of an original with respect to some purpose.

Modelle

Realität:



Modell



<https://www.flickr.com/photos/eirikref/2302281782/>

<https://www.flickr.com/photos/23465812@N00/7968397600/>

<https://creativecommons.org/licenses/by/2.0/>

Modellbegriff und SW-Modelle

- **Modell:**
- Abbildung: Welt → Diskrete Struktur.
- Vereinfachende Beschreibung der Realität.
- **Einige Aspekte, die modelliert werden:**
- Struktur.
- Beziehungen.
- Verhalten.
- Grenzen zwischen **Modell** und **Code** fließend:
kann Code als Modell auffassen.

Metamodelle

Für ein Modell M_i beschreibt ein Metamodell M_{i+1} die Sprache, in der das Modell formuliert ist.

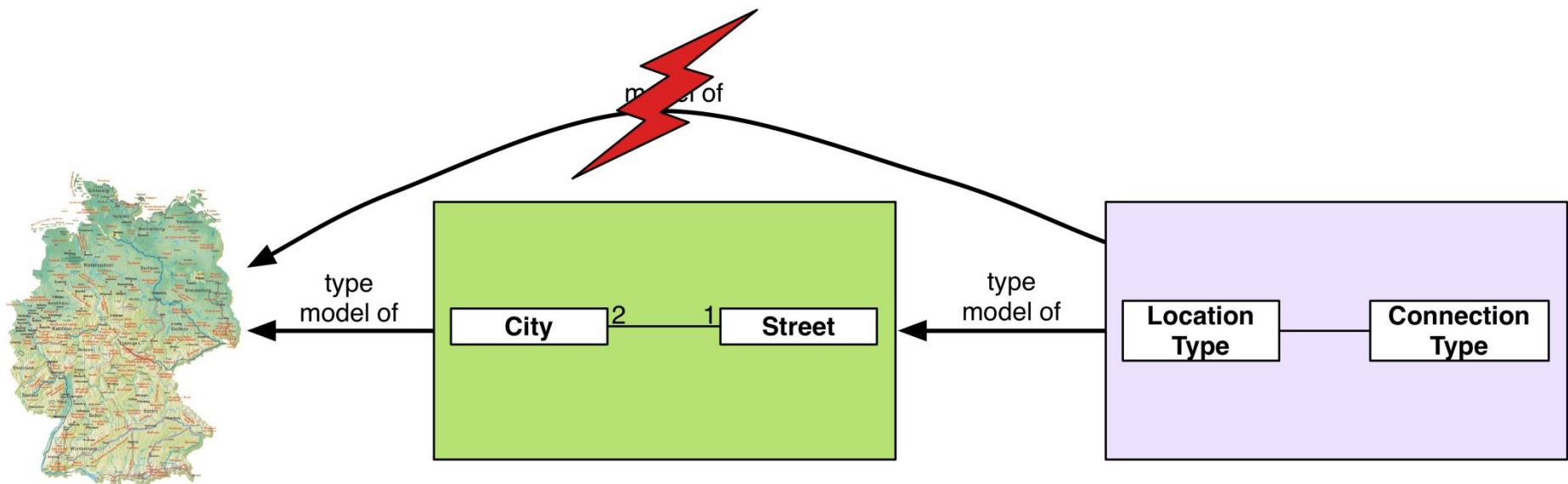
Metamodel

1. “Hence, any relation between two entities, which is going to be used to build up a meta-entity **must not be transitive.**“

2. “[...] in order to create metamodel we need **the same non-transitive relationship [...] twice.**“

Kühne, Matters of (Meta-) Modeling, STTT, Springer, 2006.

Metamodel: Model of a Model?



Yes, as we apply the same non-transitive relation “type model of“ twice!
→ anti-transitivity through double classification

Metamodellierung

Meta-Modellierung = Erstellung einer Modellierungssprache wird zum Gegenstandsbereich

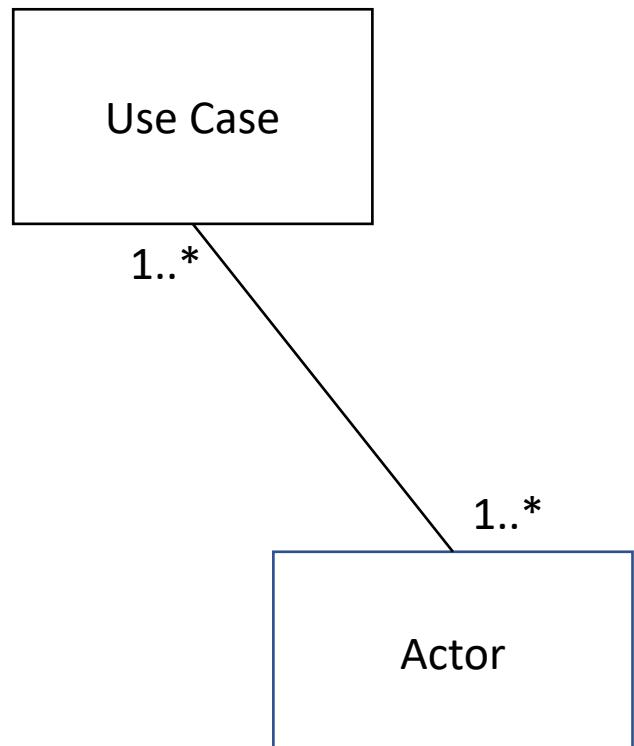
Bestandteile:

- Syntax C (konkret) und Syntax A (abstrakt)
- Semantik S
- Syntaktische Abbildung $M_s : A \rightarrow C$ (**bestimmt, wie man etwas beschreibt**)
- Semantische Abbildung $M_c : A \rightarrow S$ (**bestimmt Übersetzung**)

Semantik legt Bedeutung von Symbolen und deren Kombinationen fest.
Bedeutung definiert in begrenztem Bereich (semantische Domäne)

Abstrakte Syntax

- Definiert Struktur der Symbole durch Konzepte, Beziehungen zwischen Konzepten und Integrationsbedingungen
- Erfasst **nur** Modellierungskonzepte (Abstraktion)
- Definition der abstrakten Syntax als:
 - Graphische Sprache: Metamodelle oder Graphgrammatiken
 - Textuelle Sprache: (Baum-)Grammatiken



Meta-Informationen

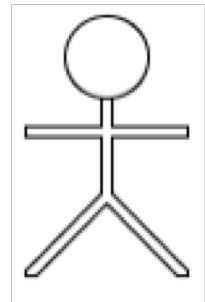
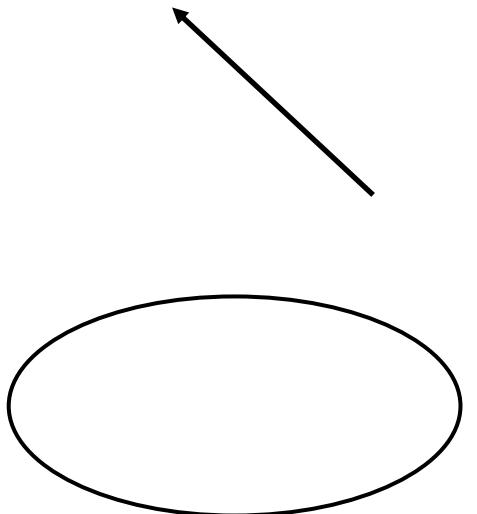
- Gültigkeitsprüfung:
 - Z.B.: darf ein Element vom Typ X mit einem Element vom Typ Y verbunden werden?
- Übersetzung:
 - Z.B. für jedes Element vom Typ X ...

Konkrete Syntax

Konkrete Syntax definiert
Symbole für
Modellierungssprache

- Textuelle Sprache
- Graphische Sprache
- Hybride (textuell und graphisch)

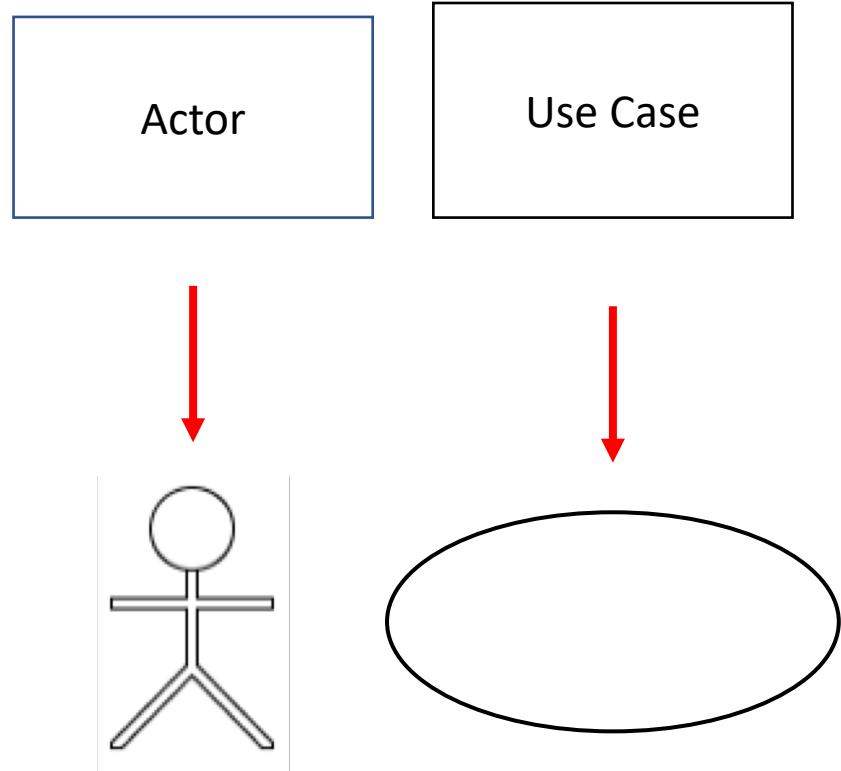
Beispiel: UML Use-Case
Diagramme



Syntaktische Abbildung

Syntaktische Abbildung: $M_s : A \rightarrow C$

Beziehung zwischen konkreter und abstrakter Syntax:



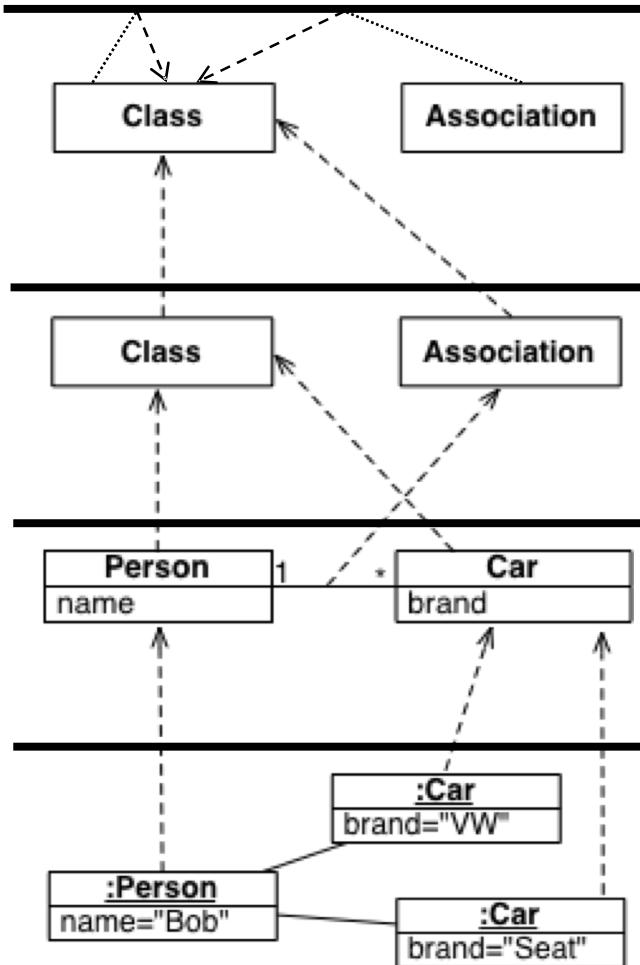
Metamodellierung in UML / MOF

Meta-Object Facility (MOF)

UML 1.x

Concrete Diagram

Concrete Objects



M3: Metametamodel

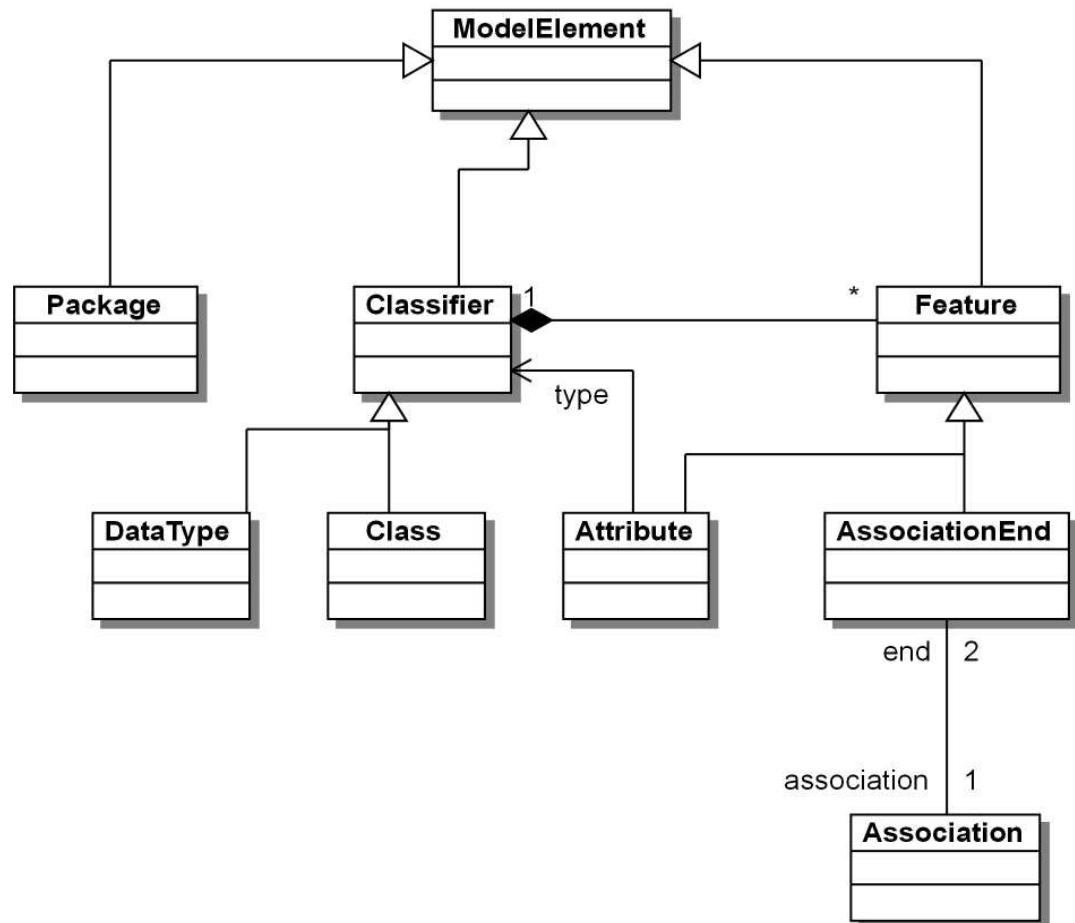
M2: Metamodel

M1: Model

M0: Model Instance

Meta Object Facility (MOF)

MOF Metamodel (vereinfacht)



Selbstreferenzierendes
Metamodell =
Metamodell
beschreibt sich selbst

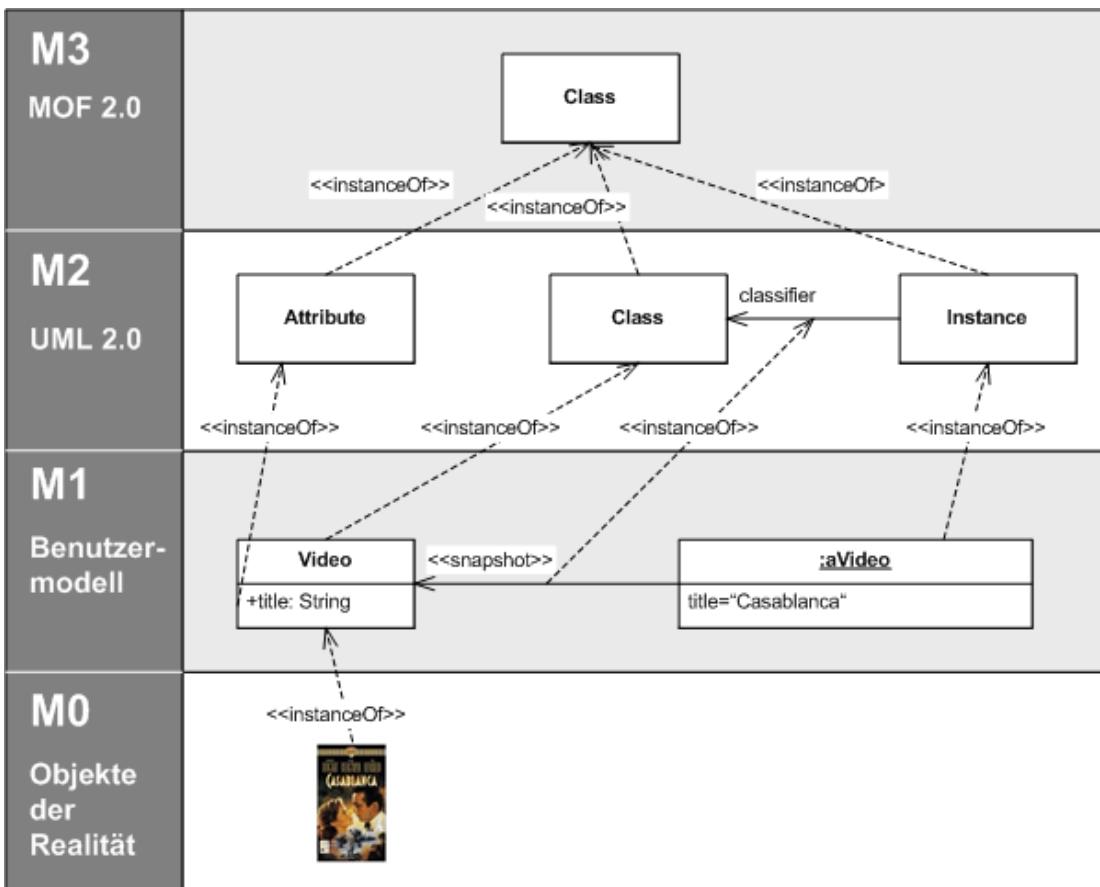
Standard MOF 2.4.2

- ISO/IEC 19508:2014
Information
technology - Meta
Object Facility (MOF)

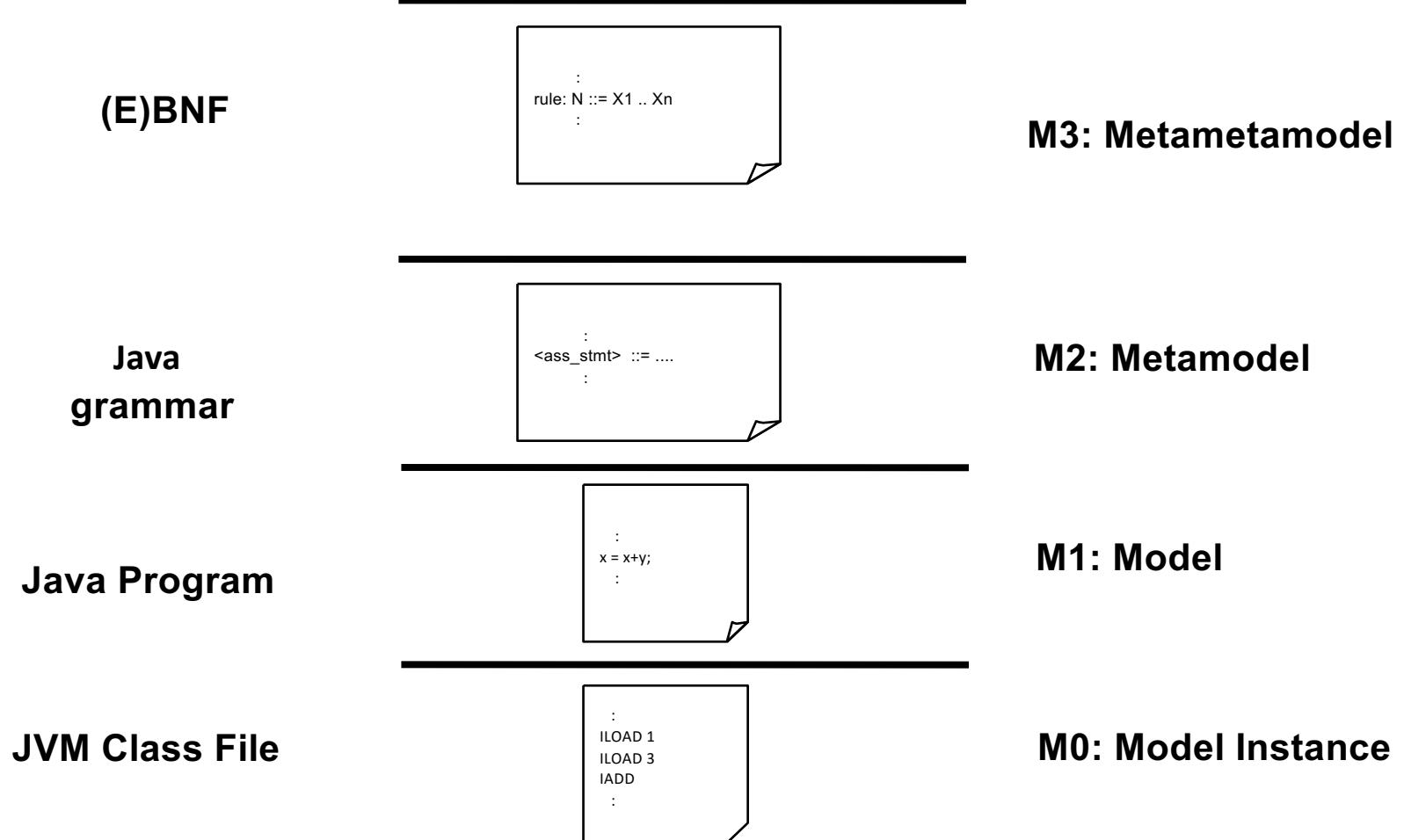
Beispiel Meta Object Facility (MOF)

Object Management Group (OMG) Standard

- M0-Ebene - Konkret
- M1-Ebene - Modelle
- M2-Ebene – Meta-Modelle
 - Definieren, wie die Modelle aufgebaut und strukturiert sind.
- M3-Ebene - Meta-Meta-Modelle (bzw. MOF-Ebene)



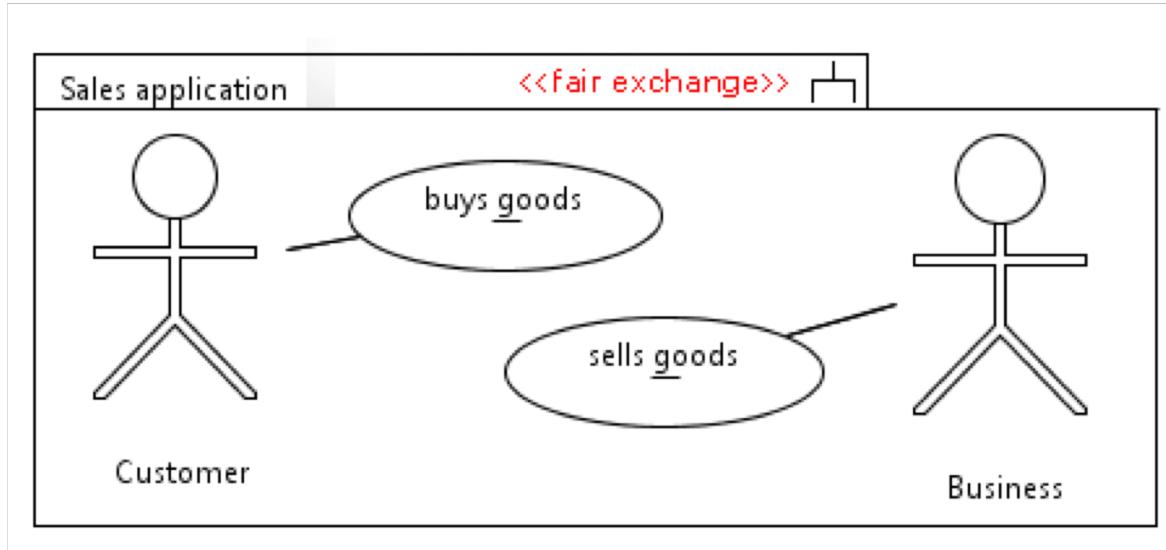
Auch Metamodellierung?



Wiederholung: Einige UML-Diagrammtypen

- **Anwendungsfalldiagramm:** Enthält die **Anforderungen** an das System
- **Klassendiagramm:** Datenstruktur des Systems
- **Aktivitätsdiagramm:** Steuerung des Ablaufs zwischen den Komponenten

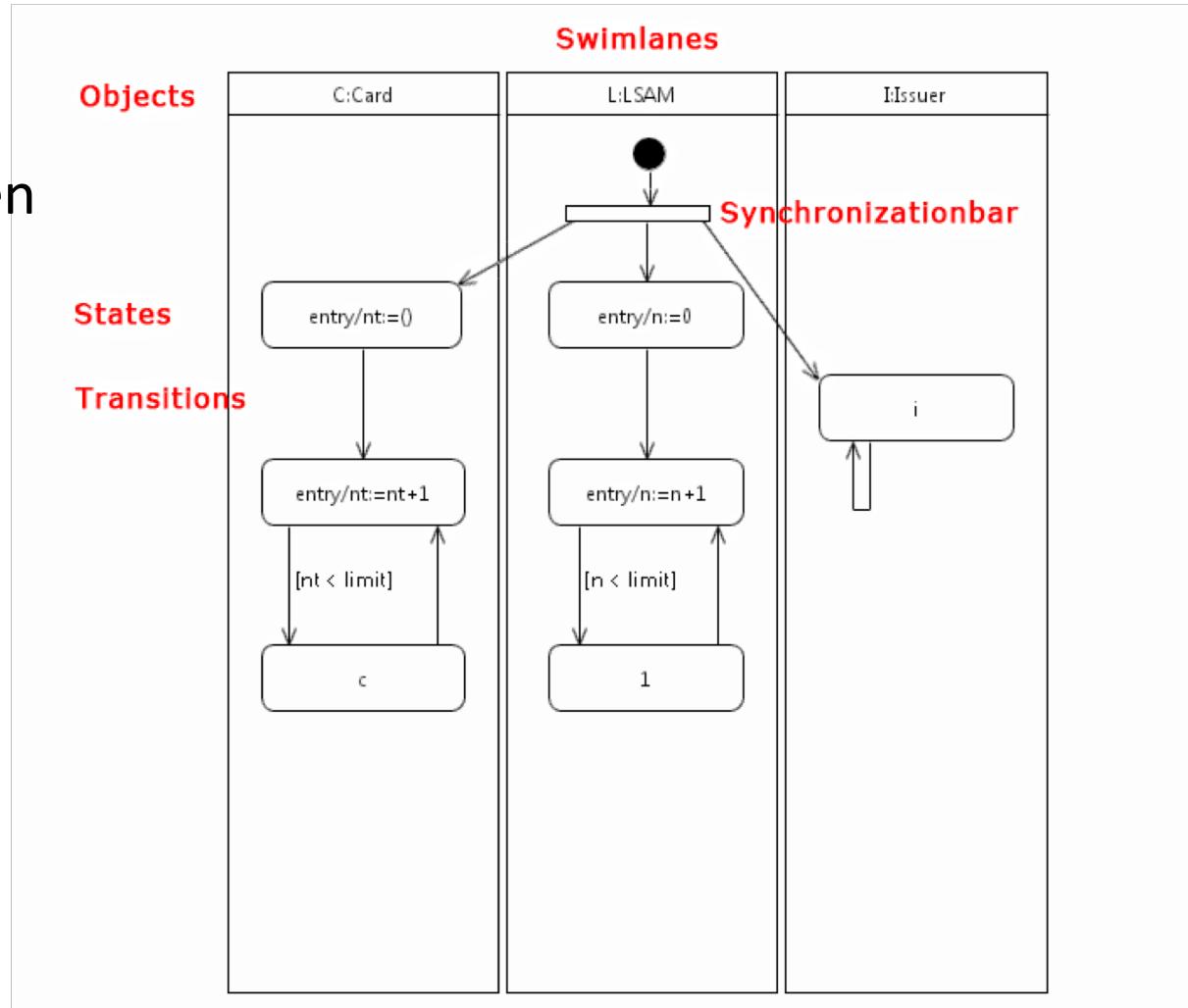
Wiederholung UML: Anwendungsfalldiagramm



- Spezifiziert einen Anwendungsfall des Systems: Szenario einer Funktionalität, die einem Benutzer oder einem anderen System angeboten wird.
- Akteure die mit einer Aktivität verknüpft sind.

Wiederholung UML: Aktivitätsdiagramm

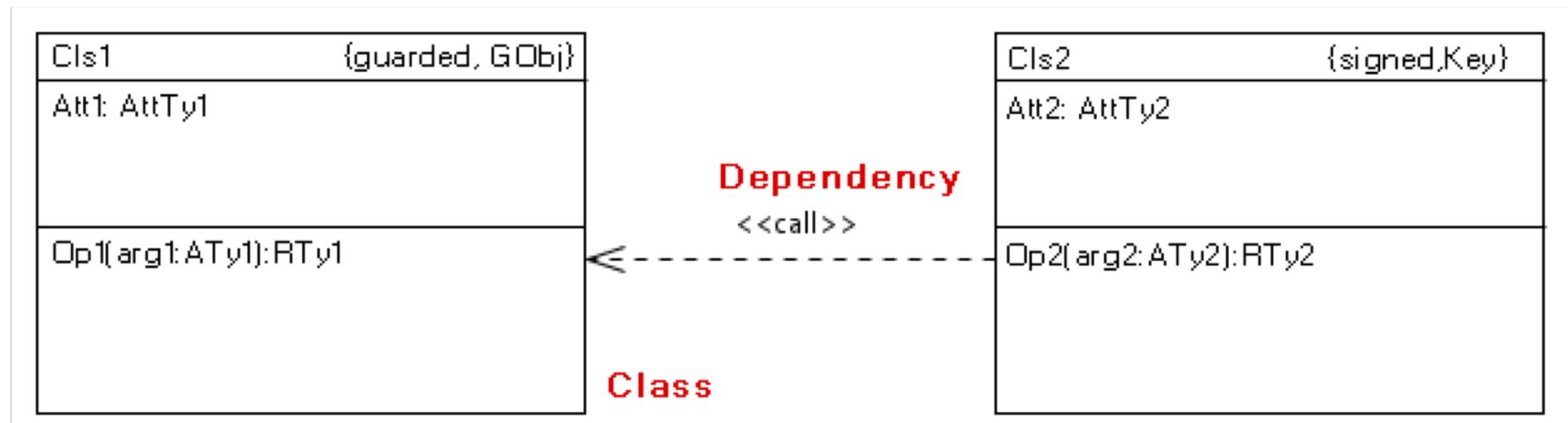
- Spezifiziert den **Kontrollfluss** zwischen Komponenten desselben Systems. Ist auf einer höheren Abstraktionsebene als Zustands- und Sequenzdiagramme.



Wiederholung UML:

Klassendiagramm

- Klassenstruktur des Systems.
- Klasse mit Attributen und Operation / Signalen, Beziehungen zwischen Klassen.

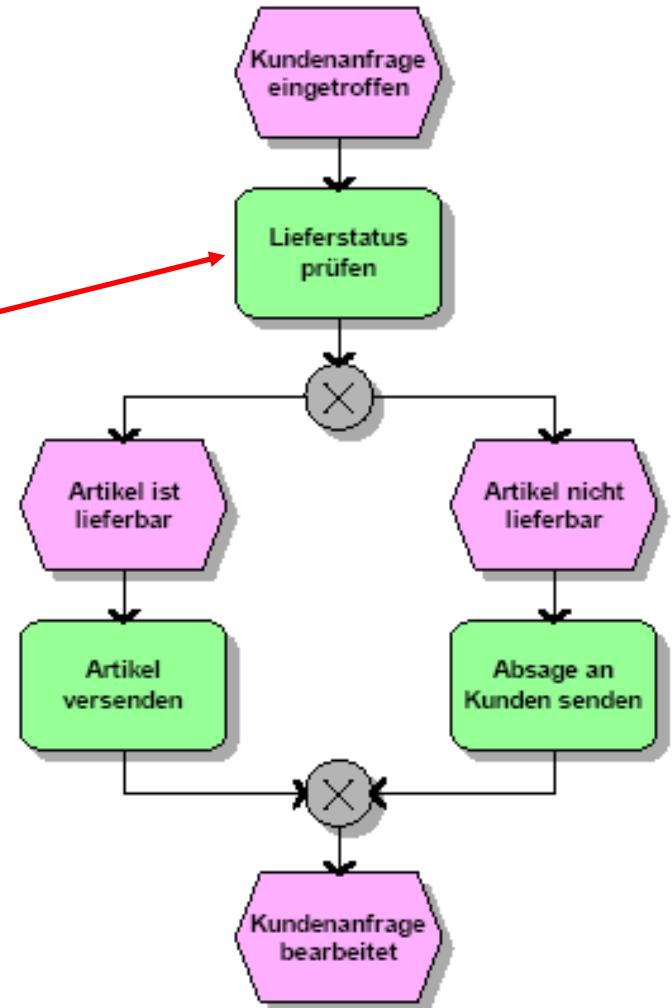


Beispiel: Event-Gesteuerte Prozessketten

Beispiel: Event-Gesteuerte Prozessketten

Anfrage in mehreren Datenbanken

Artikel aus Lager holen,
verpacken, Versandadresse laden, ...



- Modellierungssprache Prozesse
- Einzelne Schritte könnten komplexe Implementierungen haben

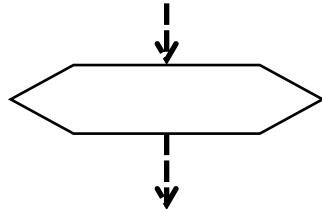
Syntax von EPKs: Ereignisse

Ereignis:

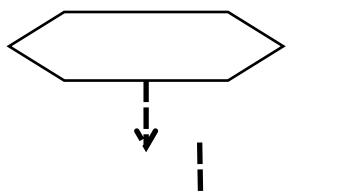
- gibt den Einstieg in einen bestimmten Zustand an.
- Beispiel:



Kontrollfluss:



Startereignis:



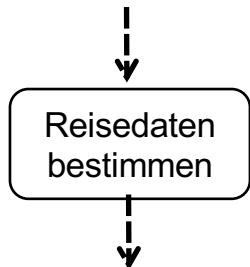
Endereignis:



Syntax von EPKs: Funktionen

Funktion:

- eine Aktivität
- Beispiel:

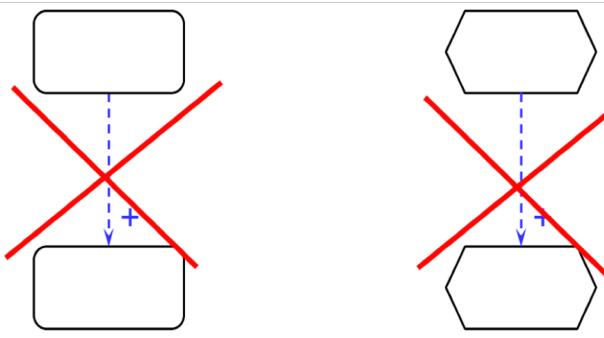
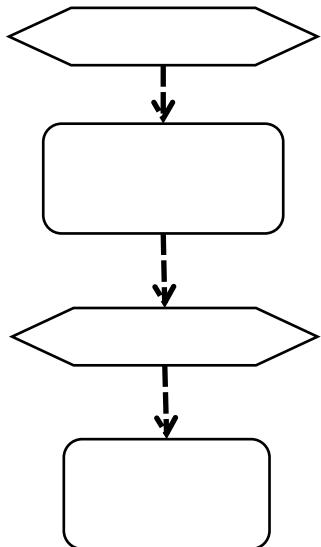


Syntax von EPKs: Kontrollfluss



Kontrollfluss:

- (nur) zwischen Ereignis und Funktion erlaubt:

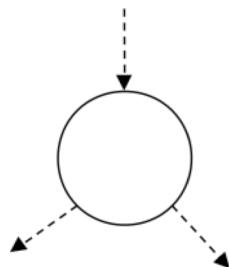


- (aber: die dafür z.T. notwendigen Trivialereignisse werden oft ausgelassen, vgl. [BMW09] S. 54).

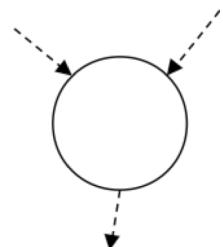
Syntax von EPKs: Konnektoren

Teilung und Verbindung des Kontrollflusses
nur an **Konnektoren**.

- **Teilung:**



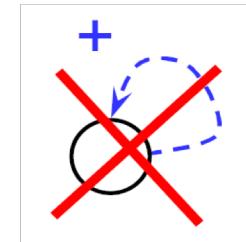
- **Verbindung:**



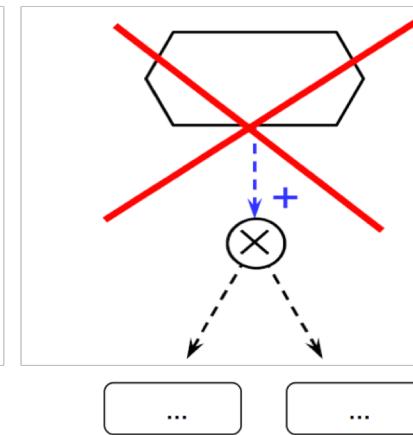
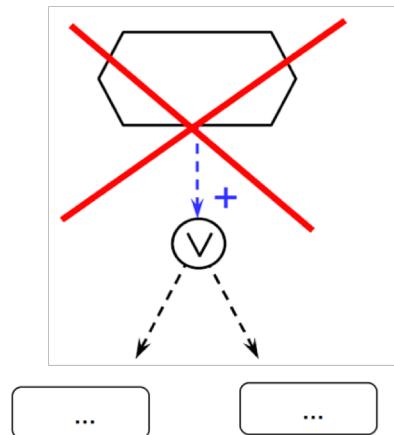
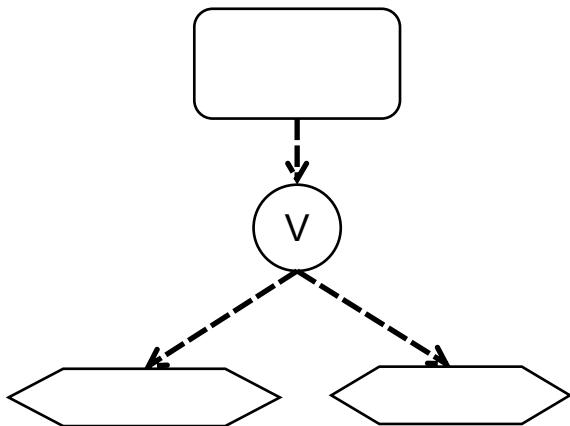
- Wobei  einer der folgenden Konnektoren:   
- Teilung und Verbindung müssen dabei zusammenpassen.

Syntax von EPKs: Konnektoren - Einschränkungen

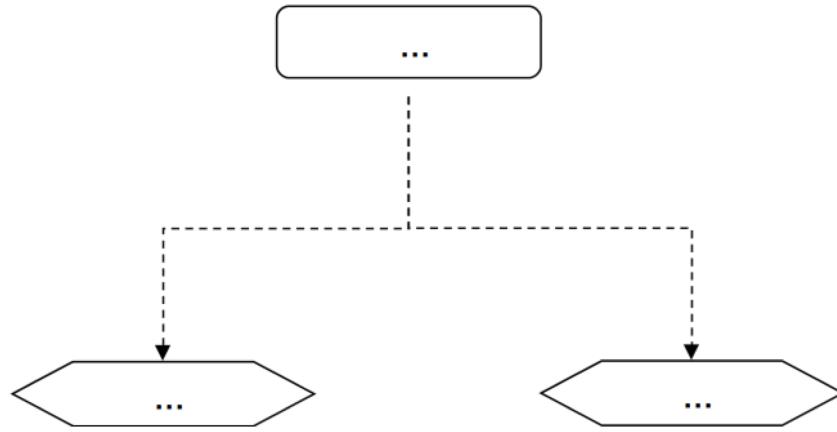
Keine Kreise an Kontrollflusskonnektoren:



- Keine (inklusive oder exklusive)
Oder-Entscheidung **nach Ereignissen**:

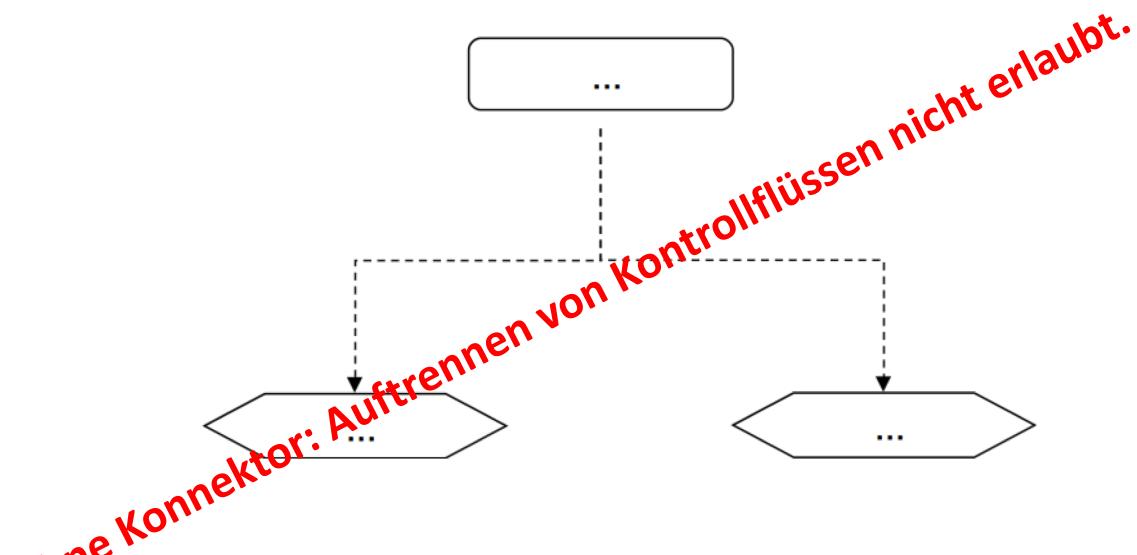


Verständnis-Frage: Kontrollfluss



Welcher Fehler liegt hier vor ?

Verständnis-Frage: Kontrollfluss

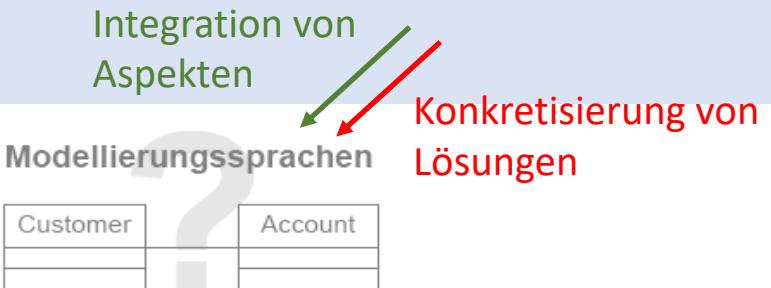


Agenda

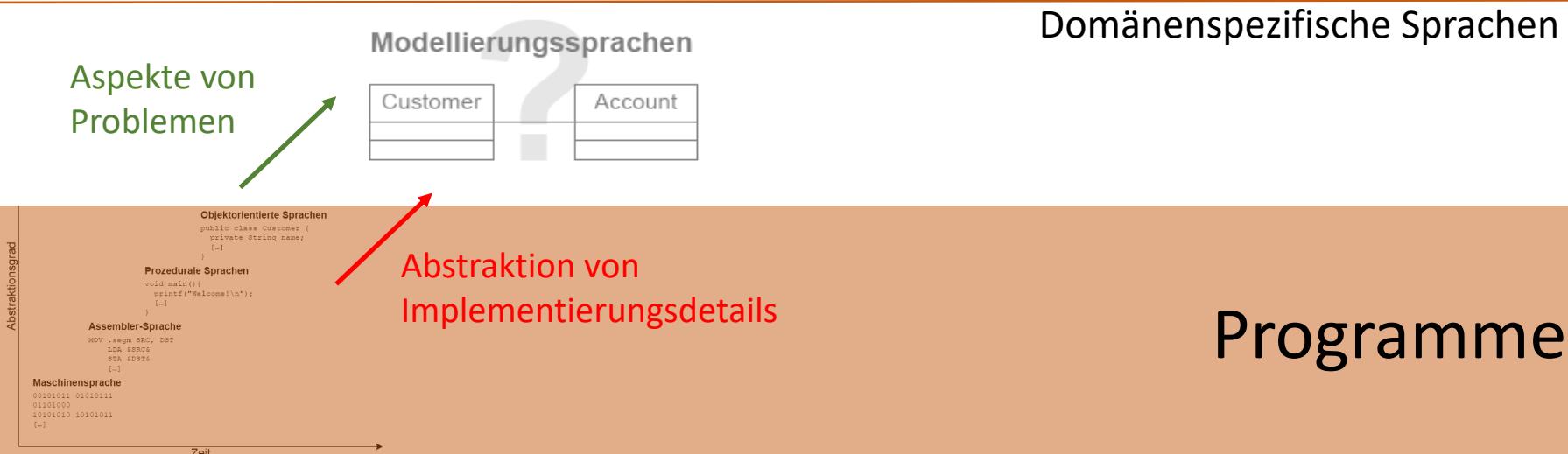
- Einleitung und Motivation
- Meta-Modellierung
- Domänen-spezifische Sprachen (=DSLs)

Von Anforderungen zu Programmen

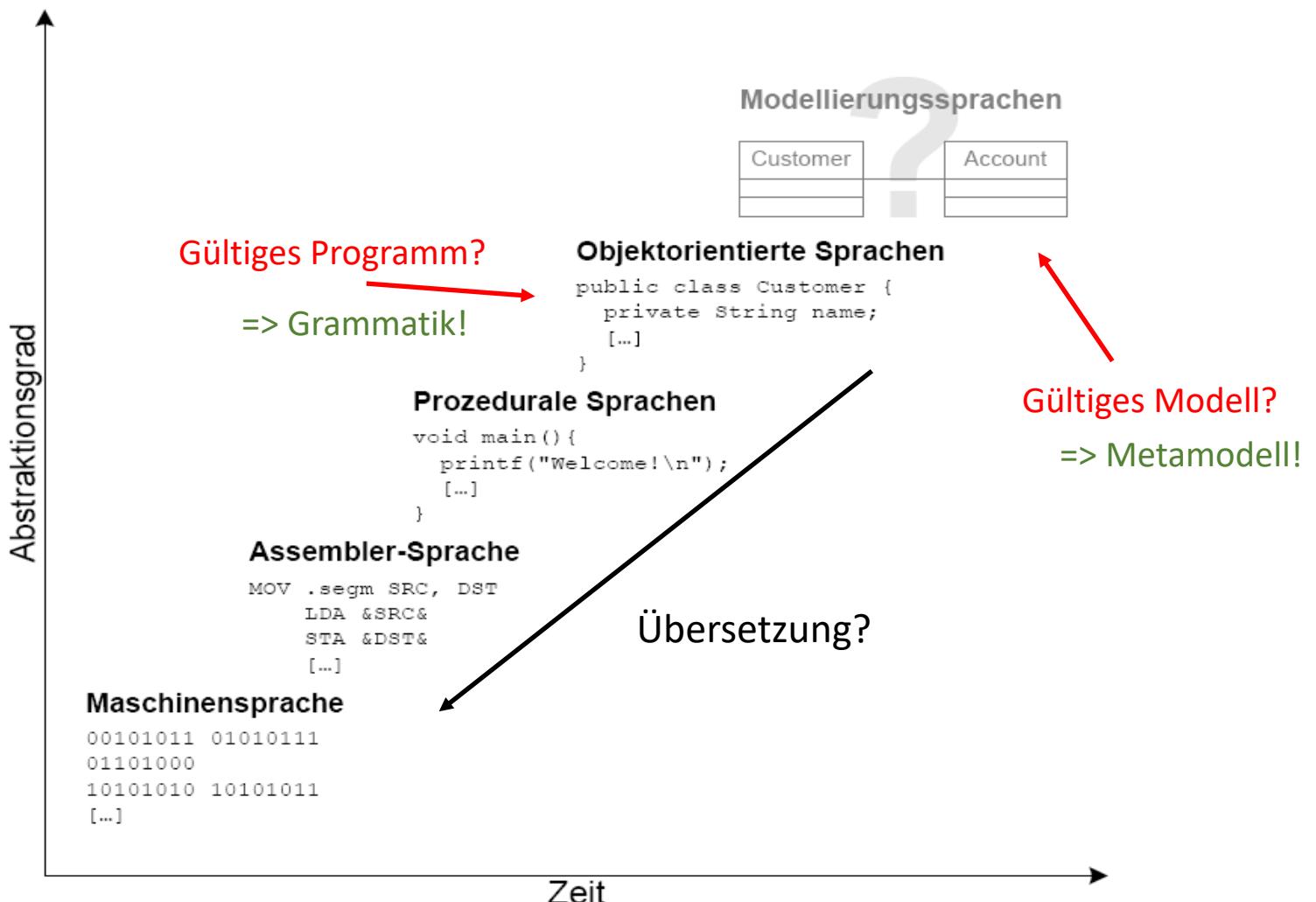
Anforderungen



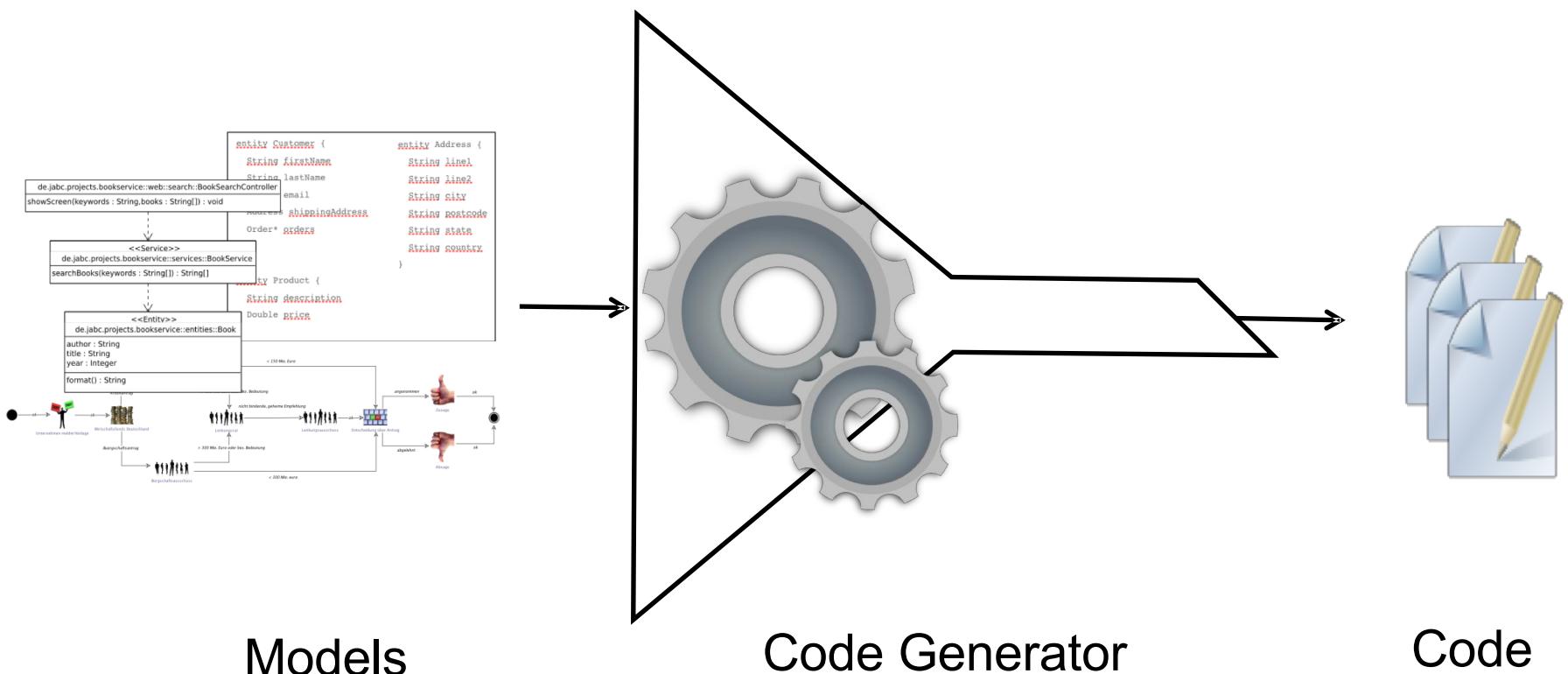
Modellgetriebene Softwareentwicklung



Motivation



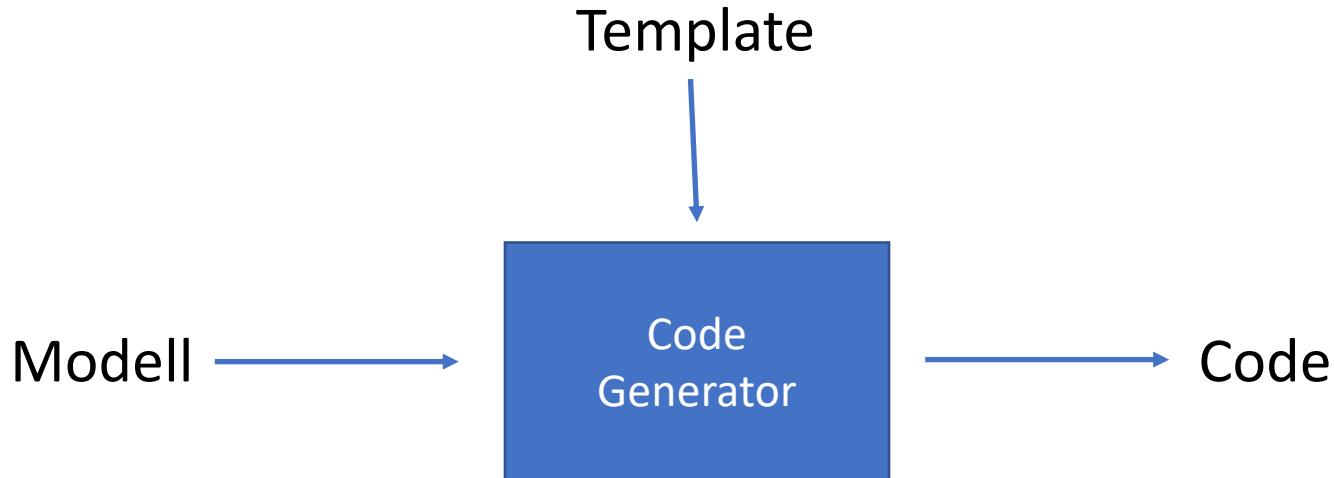
Übersetzung: Big Picture



Code Generierung

- Hier: Template-basierte Code Generierung am Beispiel erweiterter Endlicher Automaten
- Andere Verfahren in: VUC, Übersetzerbau, ...

Template-basierte Codegenerierung



Template:

- Vorlage die mit Modellinformationen ausgefüllt wird

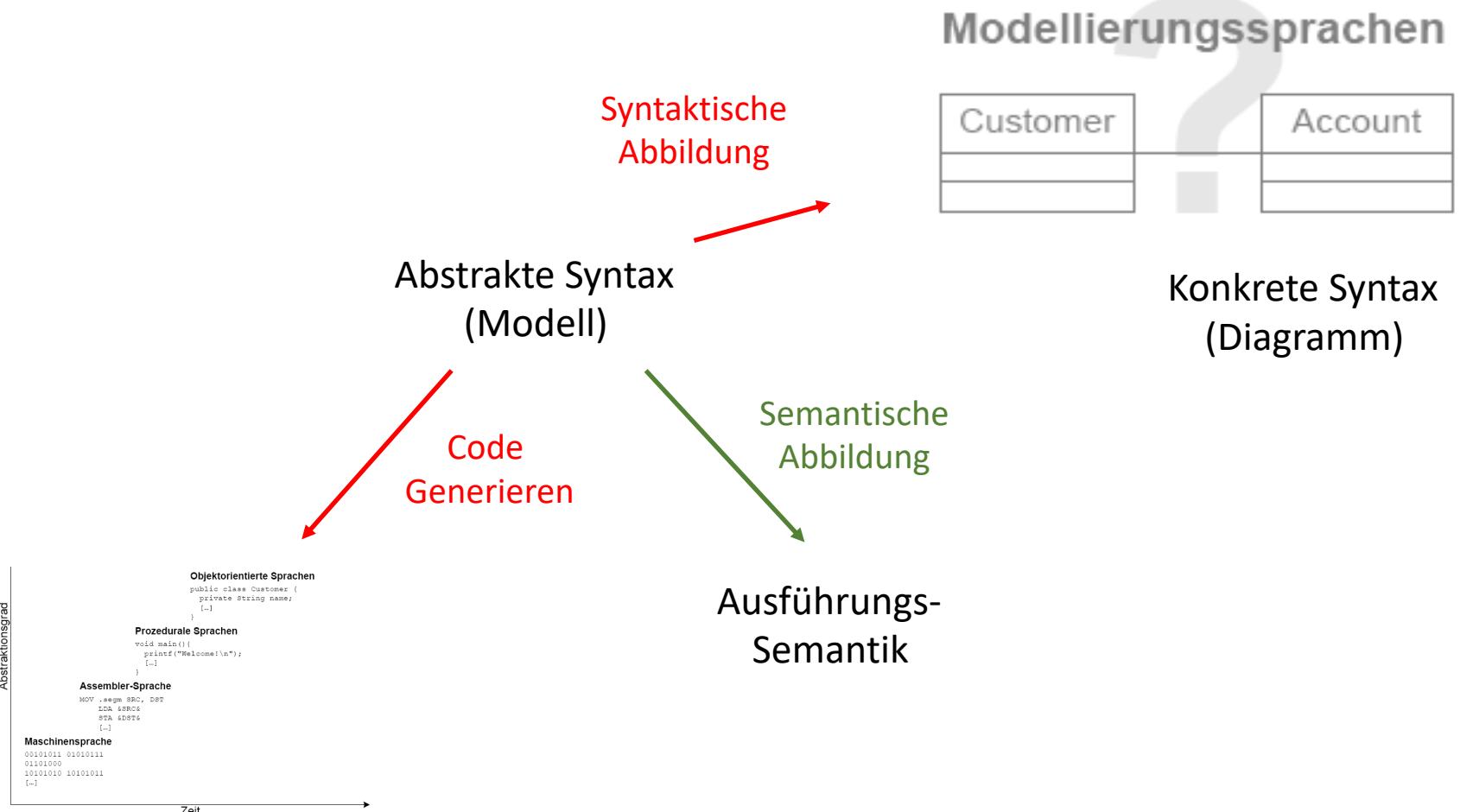
Verschiedene Templates für verschiedene Zielplattformen

Code Generator:

- Interpretiert Template Sprache und Modell

Relativ Generisch

Von Modellen zu Code



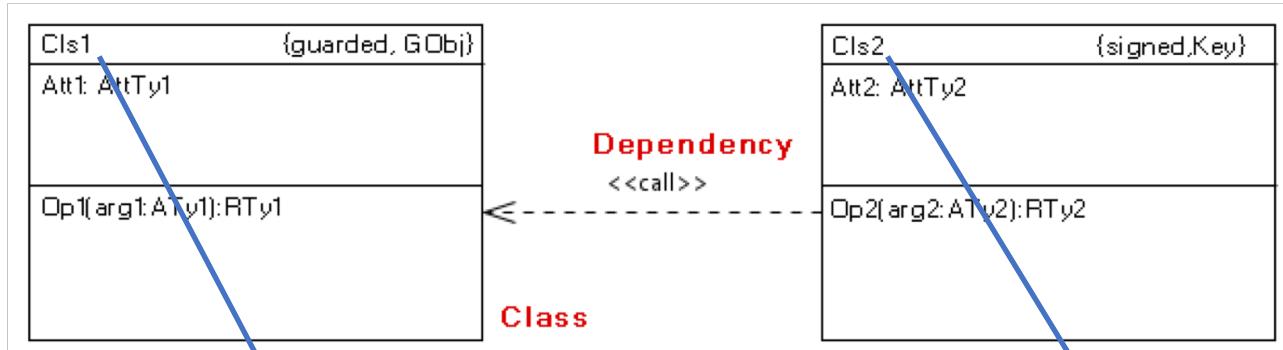
Semantische Abbildung

Übersetzung definiert durch:

- Semantische Abbildung $M_C : A \rightarrow S$
 - Semantische Domäne S
 - Z.B. Ausführungssemantik
- Beispiel
folgt später

Code Generation: Beispiel

UML Klassendiagramm



```
public class Cls1 {
    private AttTy1 Att1;
    ...
}
```

```
public class Cls2 {
    private AttTy2 Att2;
    ...
}
```

- Für alle Element vom Typ „Klassen“ ...
- Für alle Elemente vom Typ „Attribut“ ...

Zusammenfassung: Meta-Modellierung

- Metamodelle beschreiben Modelle
- Metamodelle werden benötigt, um Modelle zu automatisiert zu analysieren, zu bearbeiten oder zu übersetzen
- MOF ist das Meta Modell hinter UML
- Vier Ebenen M0 bis M3 (Reale Objekte, Modelle, Metamodelle, Metametamodelle)
- Abstrakte Syntax: Objekte und Beziehungen im Modell
- Konkrete Syntax: Darstellung im Modell