



Software Engineering

Objektorientierte Analyse

Prof. Dr. Peter Jüttner

Hochschule Deggendorf

Inhalt

5 Methoden

5.2 Architektur und Design

5.2.1 Software-Entwurf

5.2.2 Objektorientierte Analyse

5.2.2.1 Einführung und Überblick

5.2.2.2 Grundkonzepte und Elemente der UML

5.2.2.3 Statische Modellierung

5.2.2.4 Dynamische Modellierung

5.2.3 Objektorientiertes Design

5.2.4 Entwurfsmuster (Design-Pattern)

5.2.5 Modellbasierte Entwicklung

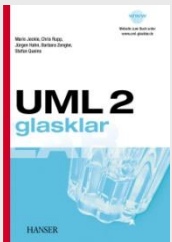
5.2.6 Architektur von Embedded Echtzeit-Systemen

5.2.7 Standard-Architekturen am Beispiel AUTOSAR

Literatur zur Vertiefung



- Helmut Balzert
Lehrbuch der Software-Technik, Teil 1
Spektrum Akademischer Verlag, 2001



- Chris Rupp, Stefan Queins, Barbara Zengler
UML 2 - Glasklar
Hanser, 2007 (3. Auflage)



- Bernd Kahlbrandt
Software-Engineering mit der UML
Springer-Verlag, 2001

Inhalt

5 Methoden

5.2 Architektur und Design

5.2.1 Software-Entwurf

5.2.2 Objektorientierte Analyse

5.2.2.1 Einführung und Überblick

5.2.2.2 Grundkonzepte und Elemente der UML

5.2.2.3 Statische Modellierung

5.2.2.4 Dynamische Modellierung

5.2.3 Objektorientiertes Design

5.2.4 Entwurfsmuster (Design-Pattern)

5.2.5 Modellbasierte Entwicklung

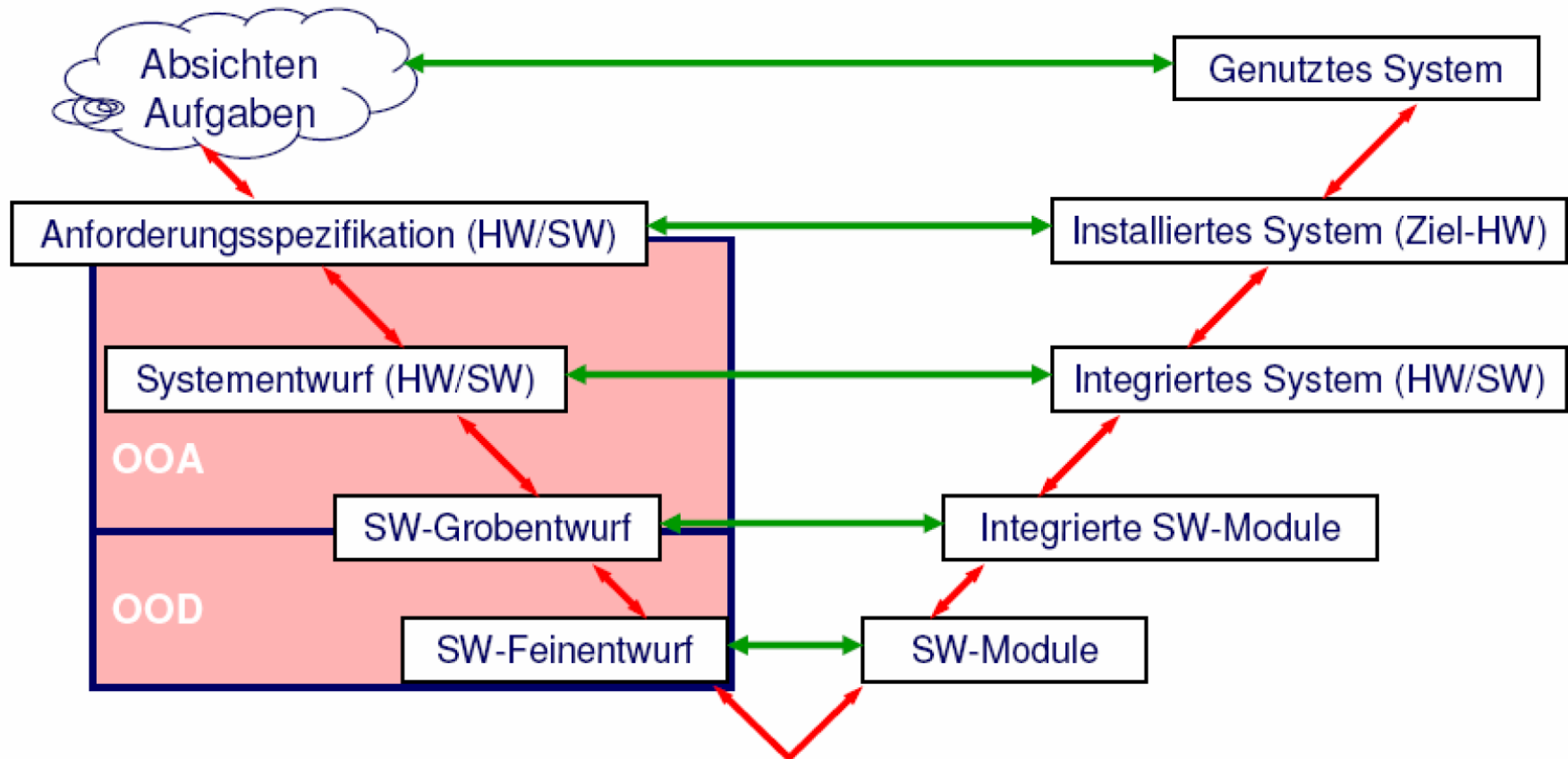
5.2.6 Architektur von Embedded Echtzeit-Systemen

5.2.7 Standard-Architekturen am Beispiel AUTOSAR

Objektorientierte Methodologie

- Die objektorientierte Methodologie wurde Ende der 80er und Anfang der 90er Jahre entwickelt.
- Die Entwicklung ging einher mit der Entwicklung objektorientierter Sprachen wie
 - Eiffel (Bertrand Meyer),
 - C++ (Bjarne Stroustrup), und
 - Objective-C (Brad Cox).
- Ziel der Objektorientierung war es, eine verbesserte Wiederverwendbarkeit und damit geringere Entwicklungskosten bei der Software-Entwicklung zu erreichen.

Bereich des V-Modells



Objektorientierte Methodologie

Die Methodologie baut auf existierende Entwurfs- und Implementierungsmethoden auf:

- **Strukturiertes Programmieren:**
 - Verwendung von Ablaufstrukturen (z.B. bedingte Anweisung und Schleifen) statt rein sequentieller Reihung von Anweisungen und Sprunganweisungen.
- **Modularisierung**
 - Lösung von Teilaufgaben in abgegrenzten Modulen bei sauberer Aufgliederung nach globalen Daten (Variablen), Übergabevariablen und lokalen Variablen.
- **Information Hiding**
 - Trennung zwischen dem Zugriff auf Information und Implementierung der Informationsverwaltung (z. B. bei relationalen Datenbanken).

Was sind Objektorientierte Modelle

- Modelle sind Abstraktionen der Realität.
- Die Kunst bei der Modell-Bildung besteht darin, alle wesentlichen Gesichtspunkte einer Problemstellung zu erfassen *und nicht Wesentliches zu vernachlässigen* (abstrahieren).
- Objektorientierte Modelle sind Abstraktionen im Bereich der Software-Entwicklung. Sie orientieren sich an den Objekten *und* den Funktionen, die zur Manipulation dieser Objekte benötigt werden.
- Diese Parallelität von Daten- und Funktionsbetrachtung ist das wichtigste Merkmal der zu Grunde liegenden Konzepte und Vorgehensweisen.

Die Kunst des Modellierens ist die
Kunst des Weglassens

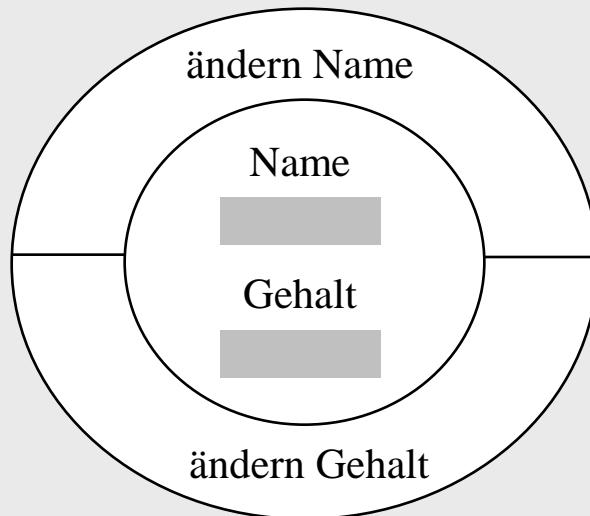
Klasse

- Eine **Klasse** ist in der objektorientierten Programmierung die Zusammenfassung
 - einer Datenstruktur und
 - der darauf anwendbaren Operationen (sog. **Methoden**) zu einer Einheit.
- Sie hat den Charakter eines Datentyps, von dem es viele Ausprägungen (sog. **Instanzen** = Objekte einer Klasse) geben kann.

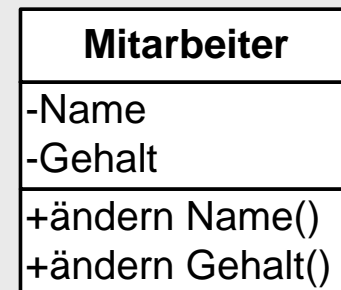
Beispiel für eine Klasse: Mitarbeiter

- Die Klasse „Mitarbeiter“ kapselt die beiden Felder „Name“ und „Gehalt“.
- Diese Felder sind nur über die Methoden „ändern Name“ und „ändern Gehalt“ änderbar.

Die Klasse "Mitarbeiter"



UML-Notation

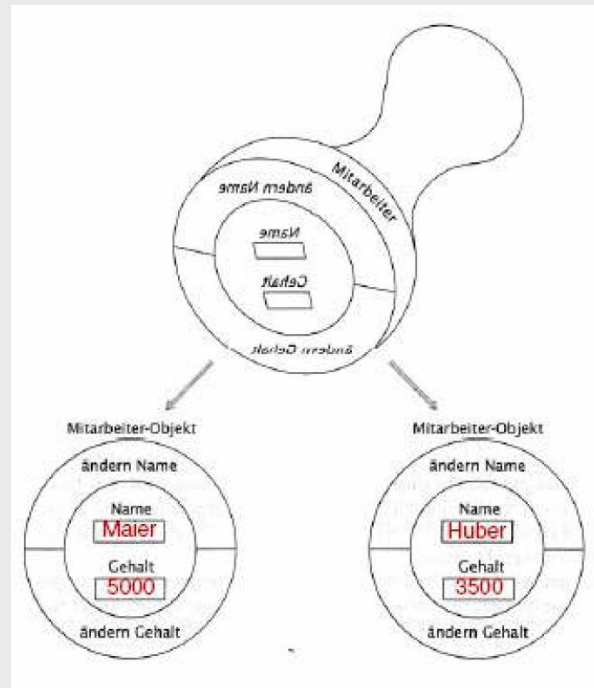


Objekt

- Ein **Objekt** ist in der objektorientierten Programmierung ein Softwaregebilde mit individuellen Merkmalen. Es definiert sich über
 - seine Identität,
 - seinen Zustand und
 - sein Verhalten„Ist was, hat was, kann was“
- Der Zustand eines Objekts ist durch die Werte der Instanzvariablen festgelegt.
- Das Verhalten eines Objekts wird durch Methoden implementiert.

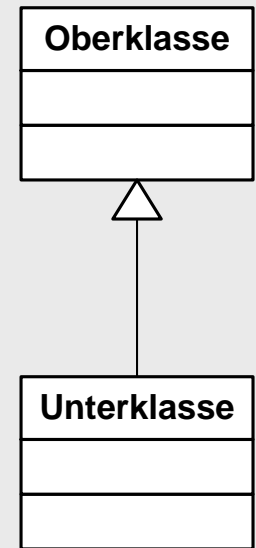
Instanzen

- In klassischen objektorientierten Sprachen sind alle Objekte **Instanzen** von Klassen.
- Die Klasse stellt also eine Schablone dar, nach der Objekte (dieser Klasse) erzeugt werden können.



Vererbung

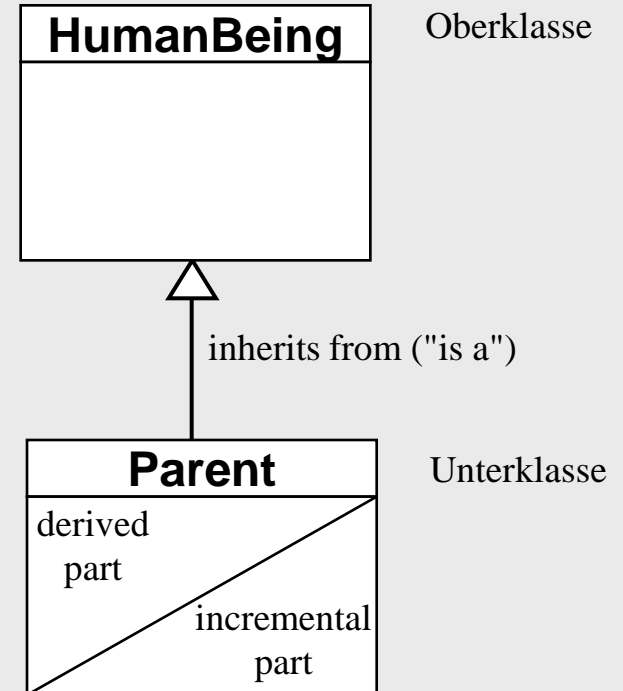
- **Vererbung** bedeutet, dass eine spezialisierte Klasse (**Unterklasse**, subclass, abgeleitete Klasse) über Eigenschaften einer oder mehrerer allgemeiner Klassen (**Oberklassen**, super classes, Basisklassen) verfügen kann.
- Eine **Unterklasse** ist vollständig konsistent mit ihrer Oberklasse bzw. ihren Oberklassen, enthält aber in der Regel zusätzliche Informationen.
- Ein **Objekt** der Unterklasse kann überall dort verwendet werden, wo ein Objekt der Oberklasse erlaubt ist
- Durch die Vererbung entsteht eine **Klassen-Hierarchie** bzw. eine Vererbungsstruktur.



Beispiel für Vererbung: Mensch/Elternteil

```
class HumanBeing
{
    private int    age;
    private float  height;
    // public declarations of operations on HumanBeing )
} // class HumanBeing

class Parent extends HumanBeing
{
    private int    numberOfChildren;
    private String  nameOfOldestChild;
    // public declarations of operations on Parent )
} // class Parent
```



Übung

Objektorientierte Analyse, Mehrfachvererbung

Beschreibung:

Mehrfachvererbung wird gelegentlich benutzt für die Darstellung von Ein-/Ausgabeverhalten mit gemeinsamen Anteilen. Dies kann eine Reihe von Problemen hervorrufen.

Aufgabe:

Überlegen Sie sich ein geeignetes Beispiel einer Klassenhierarchie mit mindestens einem Klassenattribut und einer Methode.

Erläutern Sie potentielle Probleme und deren Lösungsansätze.

Erläutern Sie auch die verschiedenen Ansätze mindestens zweier OO-Programmiersprachen (etwa C++ und Java)

Tipp:

Ein Beispiel könnte ein Input-Port, Output-Port und ein I/O-Port sein.

Zeit:

15 Minuten, arbeiten Sie ggf. zusammen mit einem Partner



Vererbung: Probleme

- Die **Änderung einer Oberklasse** kann zu Problemen führen,
 - weil die Vererbung eingesetzt wurde, um die bestehende Implementierung einer Klasse wiederzuverwenden (Inheritance for Code Reuse)
 - Weil Unterklassen Internas der Oberklasse verwenden und die Kapselung so aufgebrochen wurde.
- Vererbungsmechanismen erlauben die nahezu uneingeschränkte Redefinition von Operationen der Unterklasse, wodurch **semantische Inkompatibilitäten** zwischen redefinierter und ursprünglicher Operation entstehen können.

Polymorphie (engl.: **polymorphism**)

➤ **Grundprinzip der Polymorphie:**

- Hinter einer Objektreferenz können sich Realisierungen unterschiedlicher Objekte bzw. Methoden innerhalb derselben Klassenhierarchie verbergen.

➤ **Beispiele:**

➤ **Vererbungs-Polymorphie:**

gleiche Methoden-Bezeichnung aus der Oberklasse, unterschiedliche Semantik in verschiedenen Unterklassen

➤ **Dynamische Operator-Bindung:**

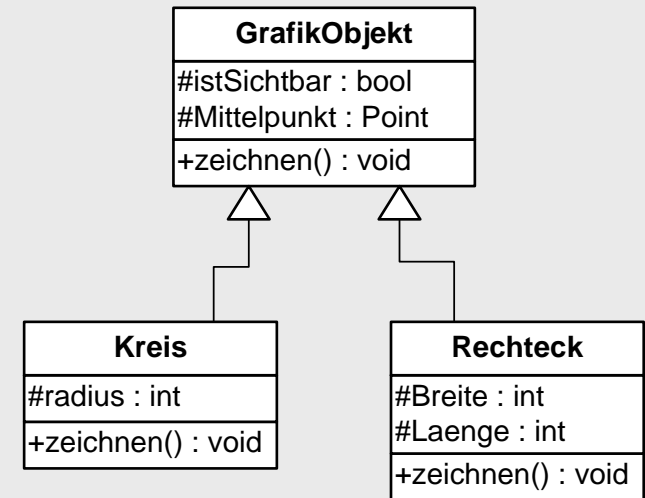
gleiche Operator-Bezeichnung,
unterschiedliche Semantik in Abhängigkeit von den Parametertypen

Späte Bindung (engl.: late binding)

- Polymorphie ermöglicht es, den gleichen Namen für gleichartige Operationen zu verwenden, die auf Objekten verschiedener Klassen auszuführen sind.
- Der Aufrufer muss nur wissen, dass das aufgerufene Objekt die gewünschte Methode besitzt. Er muss nicht wissen, zu welcher Klasse das Objekt gehört. Erst zur Laufzeit des Programms wird bestimmt, zu welcher Klasse das aufgerufene Objekt gehört.
- Man spricht daher von später oder dynamischer Bindung. Polymorphie und spätes Binden (late binding) sind untrennbar verbunden.
- Dieser Mechanismus ermöglicht es, flexible und leicht änderbare Software-Systeme zu entwickeln.

Beispiel für Polymorphie

- Gegeben sei die Deklaration:
`GrafikObjekt grafik;`
- Der Operationsaufruf `grafik.zeichnen()` kann völlig unterschiedliche Wirkungsweisen besitzen.
- Ist `grafik` eine Instanz der Klasse **Kreis**, wird die Operation `Kreis.zeichnen()` aktiviert.
- Ist `grafik` hingegen eine Instanz der Klasse **Rechteck**, dann wird `Rechteck.zeichnen()` ausgeführt.
- Es wird erst zur Laufzeit des Programms bestimmt, ob die Variable `grafik` auf ein Kreis- oder ein Rechteck-Objekt zeigt.



Übung

Objektorientierte Analyse, Polymorphie am Beispiel eines Bankkontos

Beschreibung:

Es gibt unterschiedliche Arten von Bankkonten, deren Gemeinsamkeiten und Unterschiede mit Hilfe eines Vererbungsbaumes dargestellt werden sollen. Die Bankkonten unterscheiden sich in der Art der Zinsberechnung. Einen Kontostand sowie die Möglichkeiten zur Ein- und Auszahlung hat jede Kontoart.

Aufgabe:

Erstellen Sie ein kleines Programm mit den Klassen "BankAccount", "CheckingAccount" und "SavingsAccount". Erstellen Sie unterschiedliche Methoden zur Zinsberechnung. Erstellen Sie ein Testprogramm, mit dem ein paar verschiedene Bankkonten eingerichtet werden und mittels Polymorphie die Zinsberechnung entsprechend des Kontotyps durchgeführt wird.

Tipp:

Nehmen Sie zum Sammeln der Konten eine Arraylist und führen Sie die Zinsberechnung mit einer Schleife durch.

Zeit:

30 Minuten, arbeiten Sie ggf. zusammen mit einem Partner



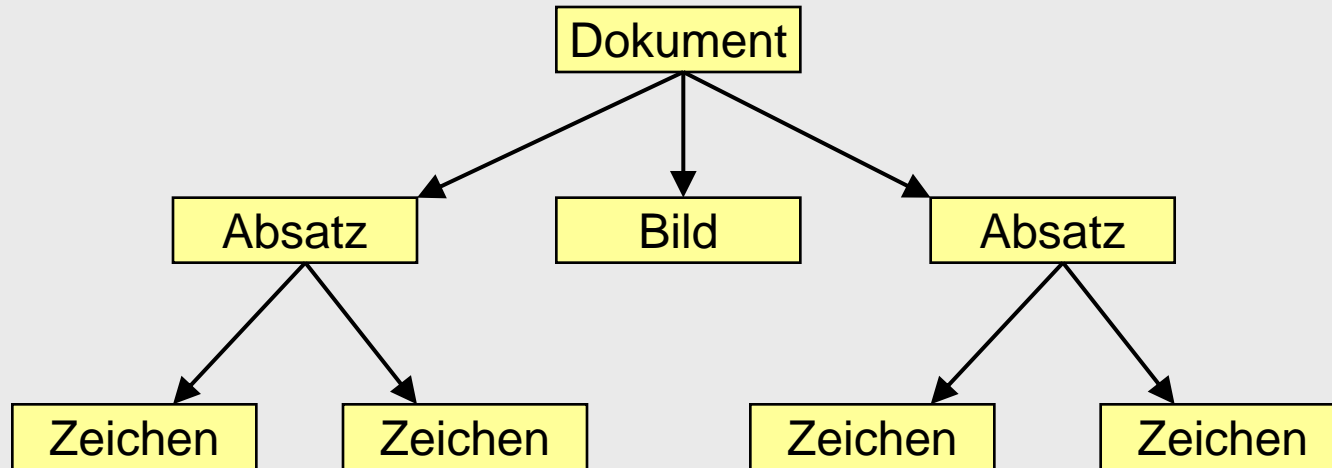
Delegation und Propagation

- Delegation ist ein Mechanismus
 - bei dem ein Objekt eine Nachricht nicht vollständig selbst interpretiert,
 - sondern an andere Objekte weiterleitet (*propagiert*), die ihrerseits einen Teil zur Ausführung der Operation beitragen
- Mittels Delegation können komplexe Strukturen auf elegante Weise vollständig traversiert werden.
(Bsp.: *Traversieren von Aggregationsbäumen*)

Delegation und Propagation

- Typische Anwendung von Propagationsmechanismen: Vollständiges Traversieren eines Aggregationsbaumes mit Start beim Wurzelobjekt, z.B. zur Realisierung von Operationen wie
 - Speichern,
 - Löschen,
 - Kopieren,
 - Ausgebendes gesamten Aggregationsbaumes.
- Voraussetzung: Alle an der Aggregation beteiligten Klassen verstehen den propagierten Operationsaufruf.

Beispiel: Speichern eines Dokumentes



Das gesamte Dokument wird wie folgt gespeichert.

1. Die Operation `save()` des Dokuments wird aufgerufen.
2. Die Operation `save()` wird an jeden Absatz und jedes Bild delegiert.
3. Die Absätze delegieren diese Operation weiter an die einzelnen Zeichen, die den Absatz ausmachen.

Inhalt

5 Methoden

5.2 Architektur und Design

5.2.1 Software-Entwurf

5.2.2 Objektorientierte Analyse

5.2.2.1 Einführung und Überblick

5.2.2.2 Grundkonzepte und Elemente der UML

5.2.2.3 Statische Modellierung

5.2.2.4 Dynamische Modellierung

5.2.3 Objektorientiertes Design

5.2.4 Entwurfsmuster (Design-Pattern)

5.2.5 Modellbasierte Entwicklung

5.2.6 Architektur von Embedded Echtzeit-Systemen

5.2.7 Standard-Architekturen am Beispiel AUTOSAR

UML – Was ist das ?

- Die Unified Modeling Language (UML) ist eine standardisierte Sprache ("Lingua Franca"), in der in allen Phasen des Software-Entwurfs (Architektur, Feindesign) Modelle methodisch erstellt und beschrieben werden können.
- UML dient zur Modellierung, Dokumentation, Spezifikation und Visualisierung komplexer Softwaresysteme
- UML ist die am meisten verbreitete Notation, um Software-Systeme zu analysieren und zu entwerfen
- Neben freiem Text sind ihre wesentlichen Elemente Grafiken
- UML bietet Notationselemente für statische und dynamische Modelle für Analyse, Architektur und Design
- UML unterstützt eine objektorientierte Vorgehensweise

UML – Wer definiert das ?

- Der UML-Standard wird von der Object Management Group (OMG) definiert
- Die OMG existiert seit 1989 mit ca. 800 Mitgliedern
- Die OMG erzeugt Industriestandards
- Die OMG ist das größte Konsortium im Software-Bereich
- Wesentliche OMG-Mitglieder:
IBM, Microsoft, Oracle, HP, Daimler, Telelogix, I-Logix...

UML – Was ist sie nicht ?

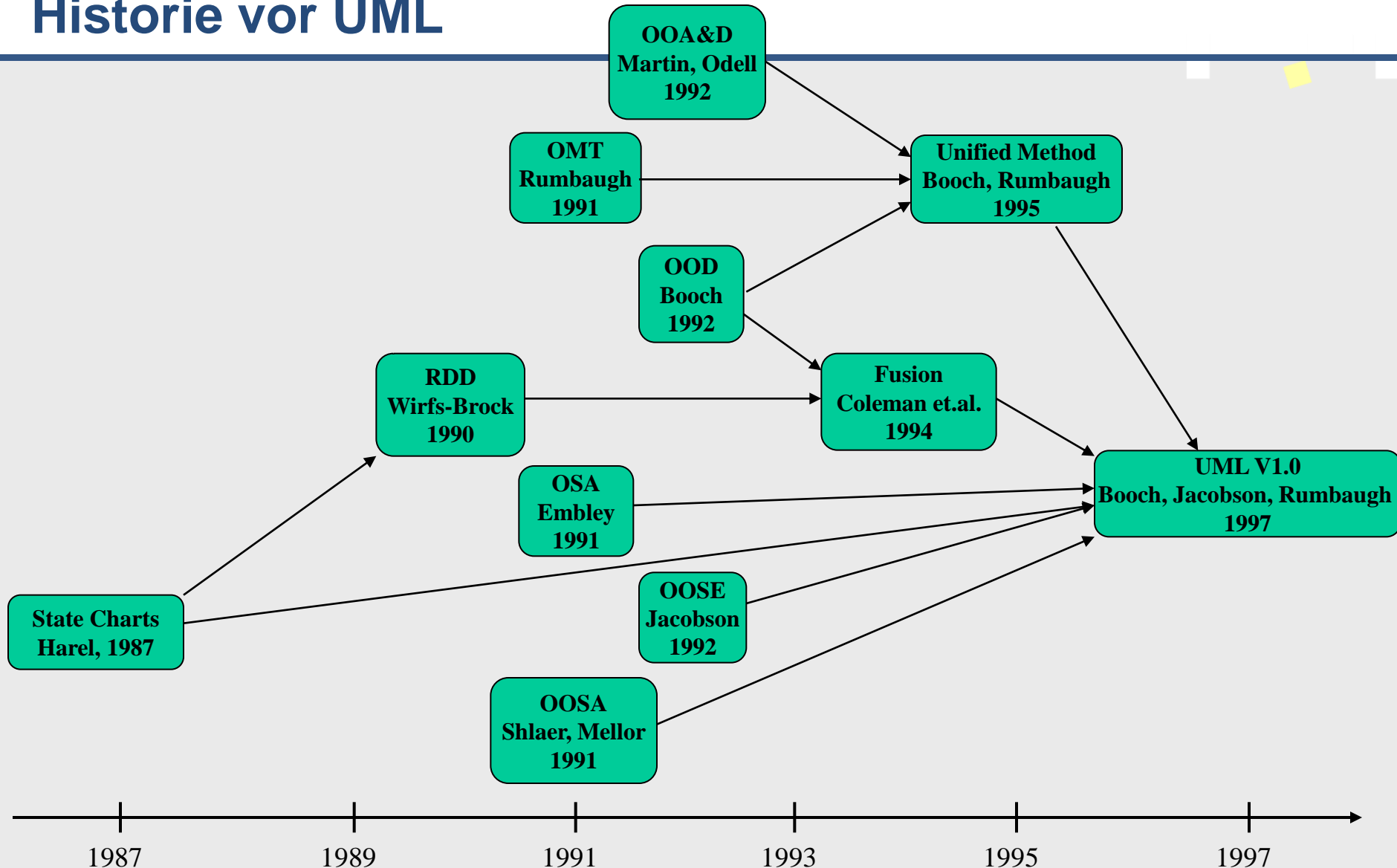
- nicht perfekt
- nicht vollständig
- keine Programmiersprache
- keine rein formale Sprache
- nicht spezialisiert auf ein Anwendungsgebiet
- kein vollständiger Ersatz für Textbeschreibung
- keine Methode, kein Vorgehensmodell

UML – Historie

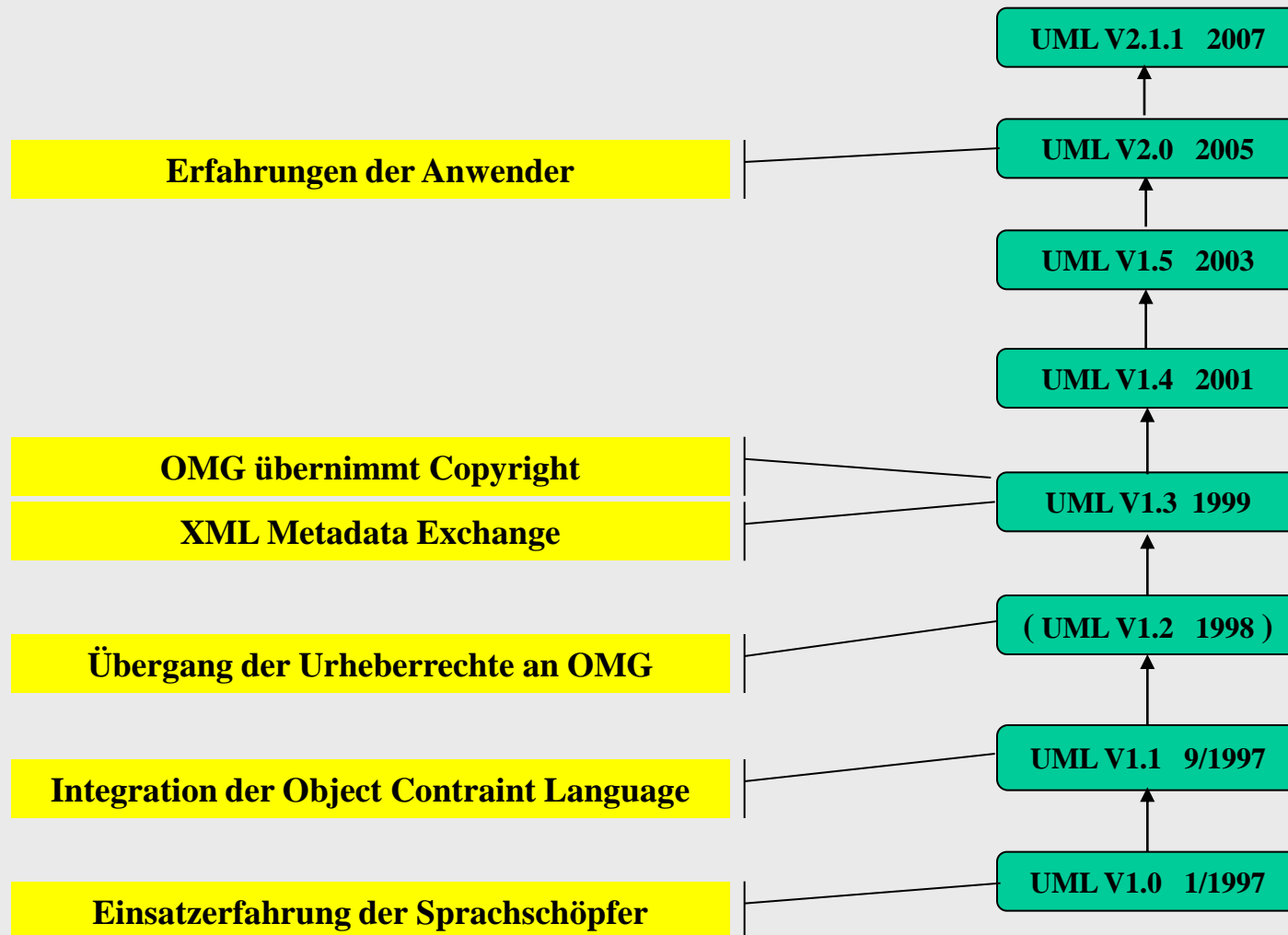
Start in den 90ern mit OO-Methoden und –Modellierungsmöglichkeiten, z.B.

- OMT von Rumbaugh ("Object Modelling Technique")
- OOD von Booch ("Object-oriented Design")
- OOSE von Jacobson ("Object-oriented Software Engineering")
- OOA & D von Martin/Odell ("Object-oriented Analysis and Design")

Historie vor UML



Historie von UML



UML 2 – Warum eine neue UML ?

Anforderungen an UML 2

- UML 1.x zu groß und komplex
- Modellierung von Systemen mit Echtzeitanwendungen
- Streichung von Sprachkonstrukten zur Vermeidung des "Second System Syndroms"
 - nicht von Toolherstellern implementiert
 - nicht in etablierten Vorgehensmodellen verwendet (z.B. parametrisierte Kollaborationen)
 - nur methoden- oder implementierungsspezifisch (z.B. Friend Stereotyp von C++)
 - Fehlen von präziser Semantik (z.B. für viele Stereotypen in UML 1.0)

UML 2 – Wege zu einer besseren UML

- **Neuformulierung des Metamodells sowie weitestgehende Verwendung der Object Constraint Language (OCL) und Wiederverwendung von Basiskonstrukten**
- **Verbesserung der Ausführbarkeit:**
 - stärkere Beziehung zwischen statischen und dynamischen Diagrammen
 - Weitere Verbesserungen durch Integration erprobter Konzepte (z.B. Petri-Netze)
- **Verbesserung der Übersichtlichkeit**
 - Verringerung der graphischen Modellkonstrukte und Basiskonzepte
 - dadurch Verbesserung der Kommunikation der Projektteilnehmer
- **Hinzufügen geeigneter Notationsmittel zur Spezifikation von Echtzeitsystemen**

UML 2 – Wege zu einer besseren UML (Fortsetzung)

- **Klarstellung von schwacher Semantik**
 - z.B. von den Beziehungen Generalisierung, Abhängigkeit und Assoziationen
- **Bessere Kapselung und Skalierbarkeit in der Verhaltensmodellierung**
 - insbesondere bei Zustandsautomaten und Interaktionsdiagrammen
- **Behebung der Einschränkungen bei der Aktivitätsmodellierung**
- **Bessere Unterstützung der hierarchischen Modellierung**
 - Systemzerlegung
 - Modellierung des "Innenlebens" von Komponenten und Funktionen
- **Bessere Unterstützung der komponentenbasierten Entwicklung (z.B. Java Enterprise Edition)**

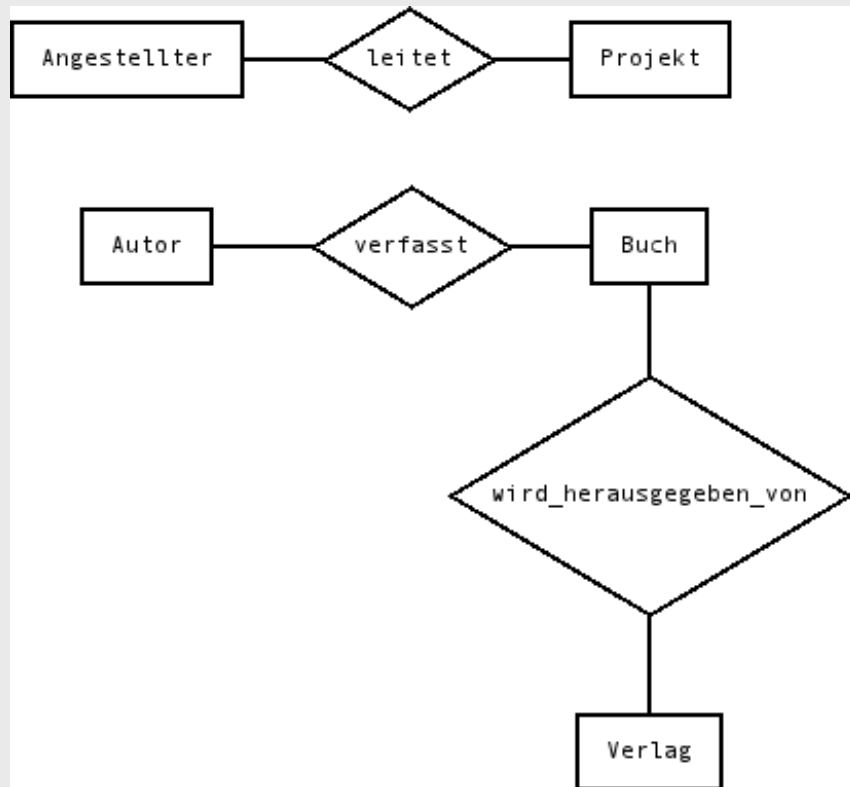
Konzept-Katalog - Überblick

- Mit der UML steht nicht nur eine standardisierte Modellierungssprache zur Verfügung, sondern auch ein vollständiger Katalog von Software-Entwicklungskonzepten.
- Die sog. Basiskonzepte gingen aus den Programmiersprachen hervor.
- Diese reichen aber nicht aus, um ein Fachmodell einer Problemstellung zu entwickeln. Es wurden deshalb Elemente der semantischen Datenmodellierung (siehe ERM!) mit aufgenommen.
- Die Synthese zwischen der Welt der objektorientierten Programmier-Sprachen und der Welt der Datenmodellierung ermöglicht die zeitunabhängigen Aspekte eines Problemfeldes zu beschreiben (Statisches Modell).
- Zur Modellierung des dynamischen Verhaltens wurden weitere Konzepte mit den zugehörigen standardisierten Notationen eingeführt und ermöglichen so auch dynamische Modelle zu entwerfen.

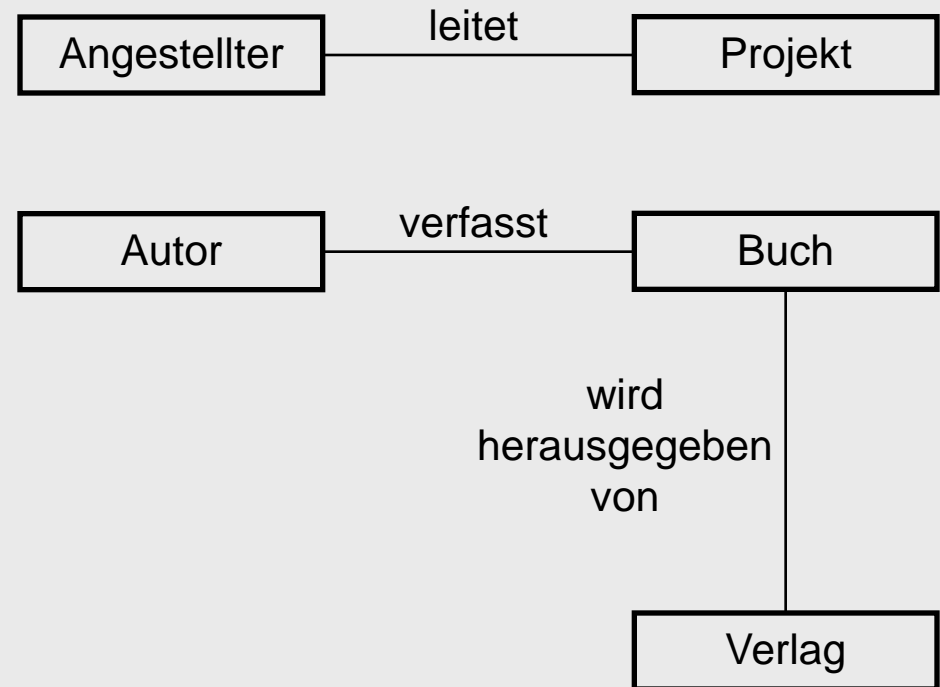
Konzept-Katalog - Überblick

➤ Beispiele für semantische Datenmodellierung

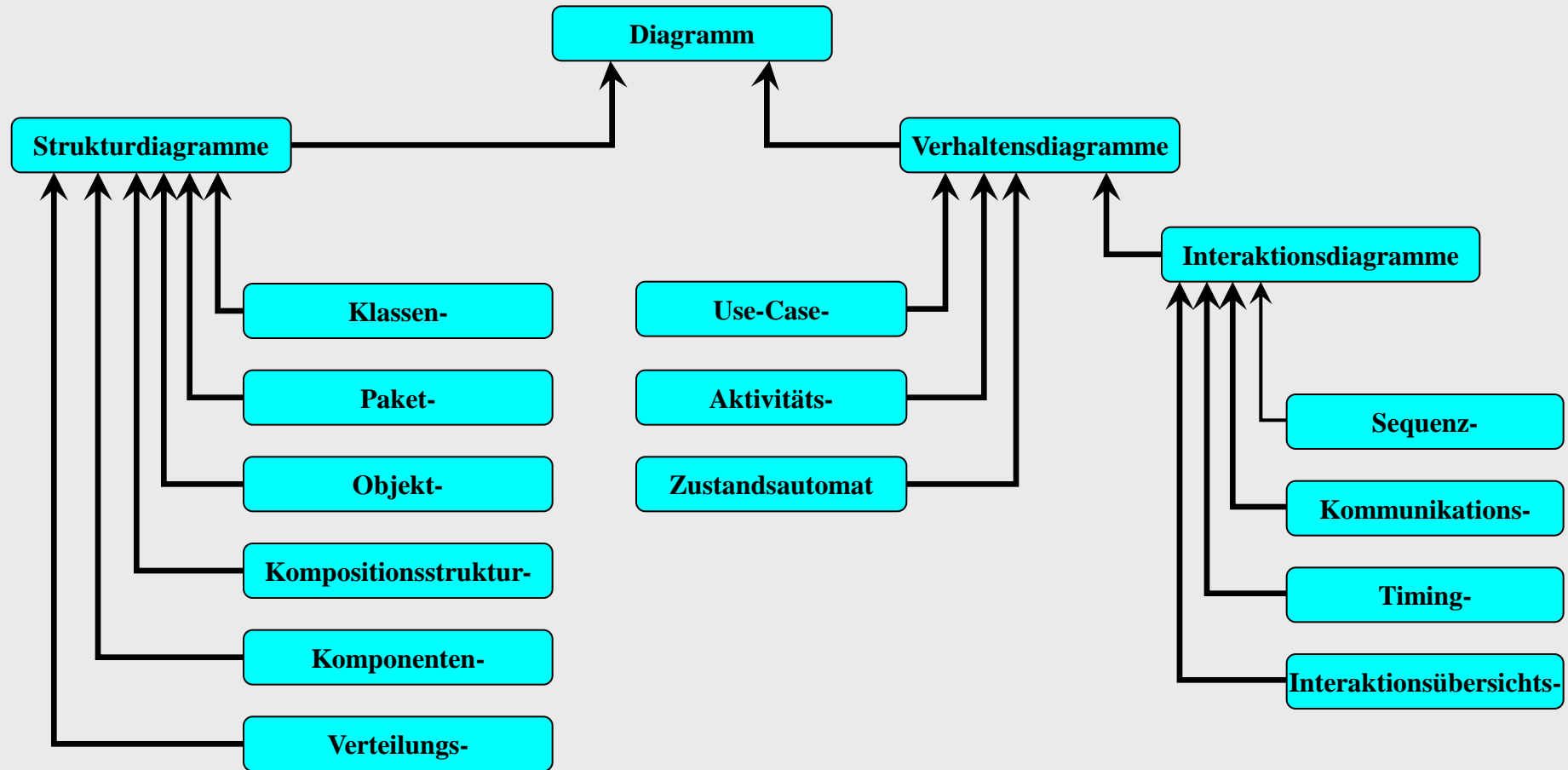
ERM



UML-Notation



Konzept-Katalog - Diagramme der UML 2



Konzept-Katalog - Statisches Modell

Datenmodellierung

Assoziation

Aggregation

Paket

Dynamische Konzepte

Use Case (Anwendungsfall)

Botschaft

Szenario

Zustandautomat

Basiskonzepte

Klasse

Objekt

Attribut

Operation

Vererbung

Polymorphismus

Konzept-Katalog - Dynamisches Modell

Datenmodellierung

Assoziation
Aggregation
Paket

Dynamische Konzepte

Use Case (Anwendungsfall)
Botschaft
Szenario
Zustandautomat

Basiskonzepte

Klasse
Objekt
Attribut
Operation
Vererbung
Polymorphismus

Diagramme der UML 2 und Ihre Anwendung

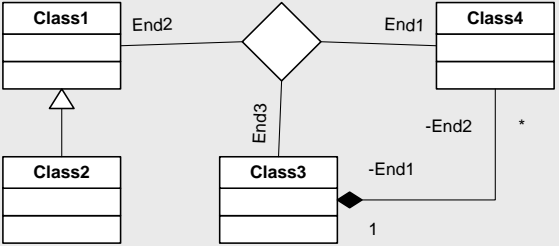
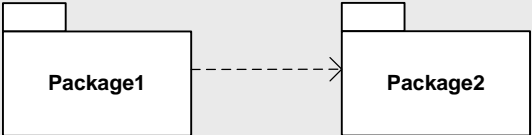
Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Klassendiagramm 	Aus welchen Klassen besteht mein System und wie stehen diese untereinander in Beziehungen	Beschreibt die statische Struktur des zu entwerfenden oder abzubildenden Systems. Enthält alle relevanten Strukturzusammenhänge und Datentypen. Bildet die Brücke zu den dynamischen Diagrammen
Paketdiagramm 	Wie kann ich mein Modell so schneiden, dass ich den besten Überblick bewahre ?	Organisiert das Systemmodell in größeren Einheiten durch logische Zusammenfassung von Modellelementen. Modellierung von Abhängigkeiten

Diagramme der UML 2 und Ihre Anwendung

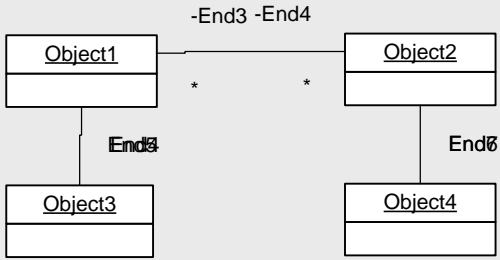
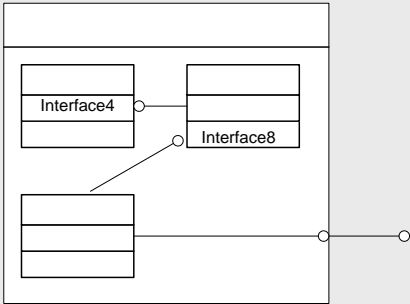
Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Objektdiagramm 	Welche innere Struktur besitzt mein System zu einem bestimmten Zeitpunkt zur Laufzeit (Klassendiagrammschnappschuß)	Zeigt Objekte und deren Attributbelegungen zu einem bestimmten Zeitpunkt. Wird nur beispielhaft zur Veranschaulichung verwendet. Detailniveau wie im Klassendiagramm. Sehr gute Darstellung von Mengenverhältnissen
Kompositionsstrukturdiagramm 	Wie sieht das Innenleben einer Klasse, einer Komponente, eines Systemteils aus ?	Ideal für die Top-down-Modellierung des Systems. Mittleres Detailniveau, zeigt Teile eines "Gesamtelements" und deren Mengenverhältnisse

Diagramme der UML 2 und Ihre Anwendung

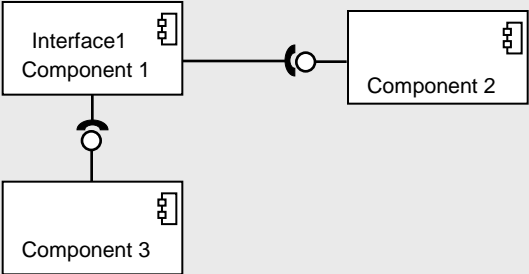
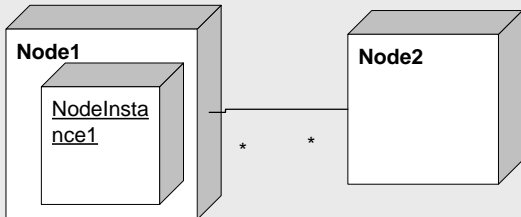
Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Komponentendiagramm 	Wie werden meine Klassen zu wieder verwendbaren, verwaltbaren Komponenten zusammengefasst und wie stehen diese miteinander in Beziehung ?	Zeigt Organisation und Abhängigkeiten einzelner technischer Systemkomponenten. Modellierung angebotener und benötigter Schnittstellen möglich.
Verteilungsdiagramm 	Wie sieht das Einsatzumfeld (Hardware, Server, Datenbanken, ...) des Systems aus ? Wie werden die Komponenten zur Laufzeit wohin verteilt ?	Zeigt das Laufzeitumfeld des Systems mit den "greifbaren" Systemteilen (meist Hardware). Hohes Abstraktionsniveau, kaum Notationselemente.

Diagramme der UML 2 und Ihre Anwendung

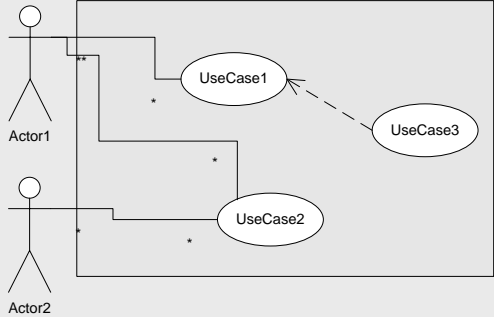
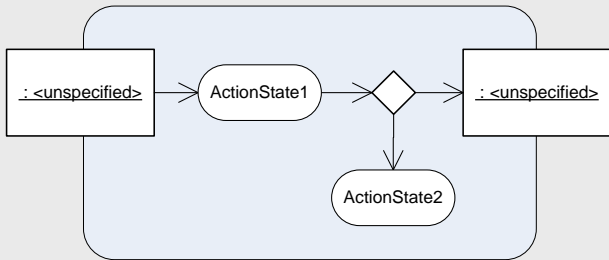
Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Use Case Diagramm 	Was leistet mein System für seine Umwelt (Nachbarsysteme, Stakeholder) ?	Präsentiert die Außensicht auf das System. Geeignet zur Kontextabgrenzung. Hohes Abstraktionsniveau, einfache Notationsmittel
Aktivitätsdiagramm 	Wie läuft ein bestimmter flussorientierter Prozess oder ein Algorithmus ab ?	Sehr detaillierte Visualisierung von Abläufen mit Bedingungen, Schleifen, Verzweigungen. Parallelisierung und Synchronisation möglich.

Diagramme der UML 2 und Ihre Anwendung

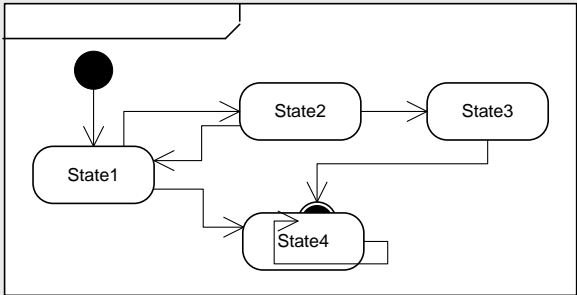
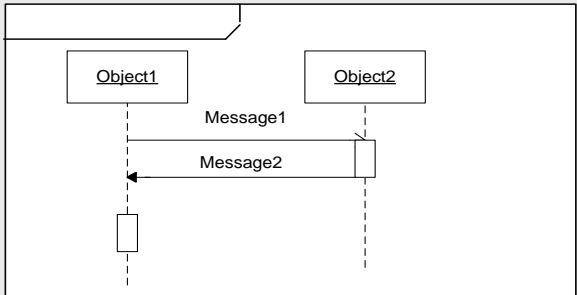
Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Zustandsautomat 	Welche Zustände kann ein Objekt, eine Schnittstelle, ein Use Case, ... bei welchen Ereignissen annehmen ?	Präzise Abbildung eines Zustandsmodells mit Zuständen, Ereignissen, Nebenläufigkeiten, Bedingungen, Ein- und Austrittsaktionen. Schachtelung möglich.
Sequenzdiagramm 	Wer tauscht mit wem welche Informationen in welcher Reihenfolge aus ?	Stellt den zeitlichen Ablauf des Informationsaustausches zwischen Kommunikationspartnern dar. Schachtelung und Flusssteuerung (Bedingungen, Schleifen, Verzweigungen) möglich

Diagramme der UML 2 und Ihre Anwendung

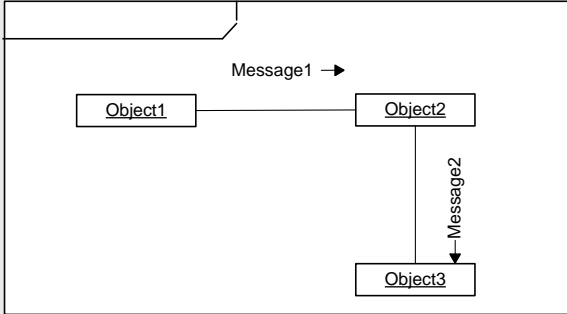
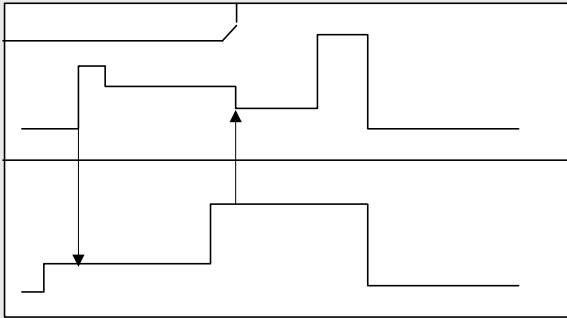
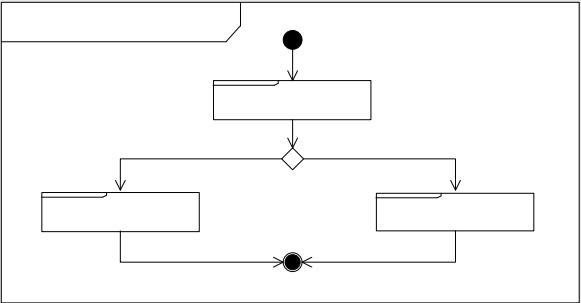
Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
Kommunikationsdiagramm  <pre> sequenceDiagram participant O1 as Object1 participant O2 as Object2 participant O3 as Object3 O1->>O2: Message1 O2->>O3: Message2 </pre>	Wer kommuniziert mit wem ? Wer "arbeitet" im System zusammen ?	Stellt den Informationsaustausch zwischen Kommunikationspartnern dar. Überblick steht im Vordergrund (Details und zeitliche Abfolge weniger wichtig)
Timingdiagramm 	Wann befinden sich verschiedene Interaktionspartner in welchem Zustand ?	Visualisiert das exakte zeitliche Verhalten von Klassen, Schnittstellen, ... Geeignet für Detailbetrachtungen, bei denen es sehr wichtig ist, dass ein Ereignis zum richtigen Zeitpunkt eintritt.

Diagramme der UML 2 und Ihre Anwendung

Diagrammtyp	Diese zentrale Frage beantwortet das Diagramm	Stärken
<p>Interaktionsübersichtsdiagramm</p>  <pre> graph TD Start(()) --> Box1[] Box1 --> Merge{ } Merge --> Box2[] Merge --> Box3[] Box2 --> End((())) Box3 --> End </pre> <p>The diagram shows a flow starting from a solid black circle (initial node) leading to a rectangular box. From this box, an arrow points down to a diamond-shaped merge node. From the diamond, two arrows branch out to two separate rectangular boxes. Both of these boxes have arrows pointing to a final bullseye node at the bottom.</p>	<p>Wann läuft welche Interaktion ab ?</p>	<p>Verbindet Interaktionsdiagramme (Sequenz-, Kommunikations- und Timingdiagramme) auf Top-Level-Ebene.</p> <p>Hohes Abstraktionsniveau.</p> <p>Gut geeignet als Strukturierung der Interaktionsdiagramme.</p>

Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalt

5 Methoden

5.2 Architektur und Design

5.2.1 Software-Entwurf

5.2.2 Objektorientierte Analyse

5.2.2.1 Einführung und Überblick

5.2.2.2 Grundkonzepte und Elemente der UML

5.2.2.3 Statische Modellierung

5.2.2.4 Dynamische Modellierung

5.2.3 Objektorientiertes Design

5.2.4 Entwurfsmuster (Design-Pattern)

5.2.5 Modellbasierte Entwicklung

5.2.6 Architektur von Embedded Echtzeit-Systemen

5.2.7 Standard-Architekturen am Beispiel AUTOSAR

Objektorientierte Analyse

- Ziel der objektorientierten Analyse (OOA) ist es,
 - geeignete Klassen zur Modellierung des Problems und
 - ihre Beziehungen untereinander zu identifizieren.
- Dies erfolgt durch
 - **statische Modellierung** (Klassendiagramm)
 - und **dynamische Modellierung** (Sequenzdiagramm, Kommunikationsdiagramm).
- Die OOA entspricht im Wesentlichen der Zielsetzung der Spezifikation und zum Teil des Grobentwurfs:
 - Antwort auf die Frage „Was tut das System?“
 - Definition des Grundgerüsts des Systems
 - Keine Antwort auf die Frage „Wie wird das Verhalten im Einzelnen realisiert?“

Statisches Systemmodell

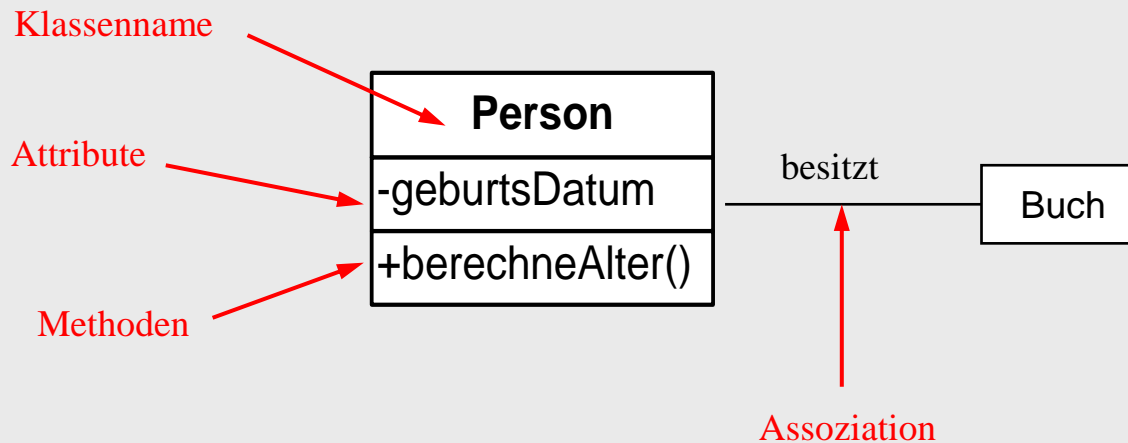
- Gegenstand des statischen Systemmodells
 - Modellierung der statischen Systemstruktur
- Ziel: Modularisieren des Systems durch
 - Identifikation der relevanten Klassen
 - Beschreibung ihrer statischen Eigenschaften (Attribute)
 - Analyse der Verantwortlichkeiten der Klassen
 - Beschreibung der Beziehungen (Assoziationen) zwischen Klassen
 - Definition der Eigenschaften der Datenspeicherung und -persistenz
- Graphische Beschreibung mit UML
 - Klassendiagramme

Klassendiagramm

- Während der OOA entsteht iterativ das **Klassendiagramm** bestehend aus den sukzessive identifizierten
 - Klassenkandidaten
 - Klassenverantwortlichkeiten
 - Assoziationen
 - Attributen
- Dieses Diagramm wird in der nachfolgenden Phase (objektorientierter Entwurf: OOD) verfeinert und ergänzt, insbesondere durch
 - Vervollständigung der Klassen
 - Beschreibung der Methoden
 - Präzisierung der Schnittstellen

Klassendiagramm

- Klassen werden durch Rechtecke dargestellt.
- Die Klassen sind zumindest mit dem Namen der Klasse beschriftet.
- Optional können Attribute und Methoden der Klasse nach dem Namen aufgeführt werden.
- Assoziationen werden durch Kanten zwischen den Klassen dargestellt.
- Optional können Assoziationen mit Namen versehen werden.



Attributdeklaration (I)

➤ **Syntax für Attributdeklaration:**

```
[Sichtbarkeit] [/] Name [:Typ] [[Multiplizität]]  
[= Vorgabewert] [{Eigenschaftswert [,  
Eigenschaftswert]*}]
```

➤ Sichtbarkeit

- public (+)
- private (-)
- protected (#)
- package (~)
- ohne Angabe: unspezifiziert, d.h. kein Default-Verhalten !

➤ / : Abgeleitetes Attribut, Wert kann zur Laufzeit berechnet werden

➤ Name: Attributname

➤ Typ: Datentyp des Attributs

Attributdeklaration (II)

- Multiplizität : $0..1, 1..1, 0..*, 1..*, n..m$
- Vorgabewert : Angabe eines festen oder berechneten Wertes
z.B.: `+pi : double = 3.1415`
- Eigenschaftswert : spezielle Eigenschaften werden in geschweiften Klammern angegeben
 - `readOnly`, z.B. `pi : Real = 3.1415 {readOnly}`
 - `ordered`
 - `unique`

Operationsdeklaration (I)

➤ **Syntax für Operationsdeklaration:**

```
<Operationsname> ::=  
[Sichtbarkeit] Name ([Parameterliste])  
[: [Rückgabetyt]  
{Eigenschaftswert [, Eigenschaftswert]*}]
```

➤ Sichtbarkeit: Definition wie bei Attributen

➤ Name : Operationsname

➤ Rückgabetyt : Datentyp des Rückgabewertes (z.B. int)

➤ Eigenschaftswert : spezielle Charakteristika der Operation

- query, nur lesender Zugriff auf Attribute
- ordered bzgl. Rückgabewerte
- unique bzgl. Rückgabewerte
- redefines <Operationsname>: Überschreiben einer geerbten Operation

Operationsdeklaration (II)

➤ **Syntax für Parameterliste:**

```
<Parameterliste> ::=  
[Übergaberichtung] Name : Typ [[Multiplizität]]  
[= Vorgabewert]  
[{Eigenschaftswert [, Eigenschaftswert]*}]
```

➤ Übergaberichtung: in, out, inout

➤ Name : **Parametername**

➤ Typ : **Datentyp des Parameters (z.B. int)**

➤ Multiplizität: 0..1, 1..1, 0..*, 1..*, n..m

➤ Vorgabewert: **Angabe eines festen oder berechneten Wertes**

➤ Eigenschaftswert: **wie bei Attributen**

Schritte zum Statischen Modell

1. Klassenkandidaten identifizieren
2. Assoziationen identifizieren
3. Attribute spezifizieren

Klassenkandidaten identifizieren

- Wenn man sich in einem Anwendungsbereich auskennt, wird man schnell eine ganze Reihe von Klassenkandidaten nennen können.
- Sehr viel schwieriger wird es aber sein, exakt zu begründen, warum man gerade auf diese kommt.
- Folgende beispielhafte Vorgehensweise zeigt, wie man sinnvolle Klassen- bzw. Attributskandidaten erkennen kann.
- Erste Kandidaten für Klassen ergeben sich aus den Anforderungen, in dem man alle **Substantive** betrachtet.

Beispiel: Bibliothek

- Produktbeschreibung (Auszug):
 - Eine Bibliothek besitzt Exemplare von Büchern, Zeitschriften etc.
 - Die meisten Buchexemplare können ausgeliehen werden. Nicht ausgeliehen werden können so genannte Präsenzexemplare von Büchern. Zeitschriften können ebenfalls nicht ausgeliehen werden.
 - Sind von einem Buch alle Exemplare ausgeliehen, so kann es vorgemerkt werden. Wird ein Exemplar eines vorgemerkten Buches zurückgegeben, so wird der erste Vorbesteller benachrichtigt. Holt er das Buch nicht binnen einer Woche ab, so verfällt die Vormerkung.
 - Wird die Leihfrist überschritten, so wird der Benutzer gemahnt. Er wird solange von der Ausleihe ausgeschlossen, bis das Exemplar zurückgegeben wird.

Beispiel: Bibliothek

- Produktbeschreibung (Auszug):
 - Eine **Bibliothek** besitzt **Exemplare** von **Büchern**, **Zeitschriften** etc.
 - Die meisten **Buchexemplare** können ausgeliehen werden. Nicht ausgeliehen werden können so genannte **Präsenzexemplare** von **Büchern**. **Zeitschriften** können ebenfalls nicht ausgeliehen werden.
 - Sind von einem **Buch** alle **Exemplare** ausgeliehen, so kann es vorgemerkt werden. Wird ein **Exemplar** eines vorgemerkten **Buches** zurückgegeben, so wird der erste **Vorbesteller** benachrichtigt. Holt er das **Buch** nicht binnen einer **Woche** ab, so verfällt die **Vormerkung**.
 - Wird die **Leihfrist** überschritten, so wird der **Benutzer** gemahnt. Er wird solange von der **Ausleihe** ausgeschlossen, bis das **Exemplar** zurückgegeben wird.

Beispiel für Klassenkandidaten: Bibliothek

- Betrachtet werden zunächst die Substantive, die in der Beschreibung und in den Anwendungsfällen vorkommen.
- Das gibt Anlass zu folgenden Klassenkandidaten (in alphabetischer Reihenfolge):

Ausleihe	Benutzer	Bibliothek
Buch	Buchexemplar	Exemplar
Leihfrist	Präsenzexemplar	Vorbesteller
Vormerkung	Woche	Zeitschrift
- Diese müssen nun daraufhin untersucht werden, ob und welche Rolle sie im Anwendungsbereich spielen. Auf dieser Basis kann dann entschieden werden, ob sich Klassen aus ihnen ergeben.

Beispiel für Klassenauswahl: Bibliothek

➤ Klassenkandidaten durch Namensextrahierung

Ausleihe	Benutzer	Bibliothek
Buch	Buchexemplar	Exemplar
Leihfrist	Präsenzexemplar	Vorbesteller
Vormerkung	Woche	Zeitschrift

➤ Ordnen nach Zusammengehörigkeit:

- Buch, Buchexemplar, Exemplar, Präsenzexemplar, Zeitschrift
- Vormerkung
- Ausleihe, Leihfrist (Konstante), Woche (Konstante)
- Benutzer, Vorbesteller
- Bibliothek (System selbst ist keine Klasse)

Beispiel für Klassenauswahl: Bibliothek

- Nach Entfernen der Konstanten und des Gesamtsystems bleiben übrig:
 - Buch, Buchexemplar, Exemplar, Präsenzexemplar, Zeitschrift
 - Vormerkung
 - Ausleihe
 - Benutzer, Vorbesteller
- Aufgrund der Problembeschreibung sind Zeitschriften spezielle Präsenzexemplare, die keine eigene Klasse erfordern.
- Buchexemplar ist ein Synonym von Exemplar.
- Vorbesteller sind Benutzer in einer besonderen Rolle.

Assoziationen identifizieren

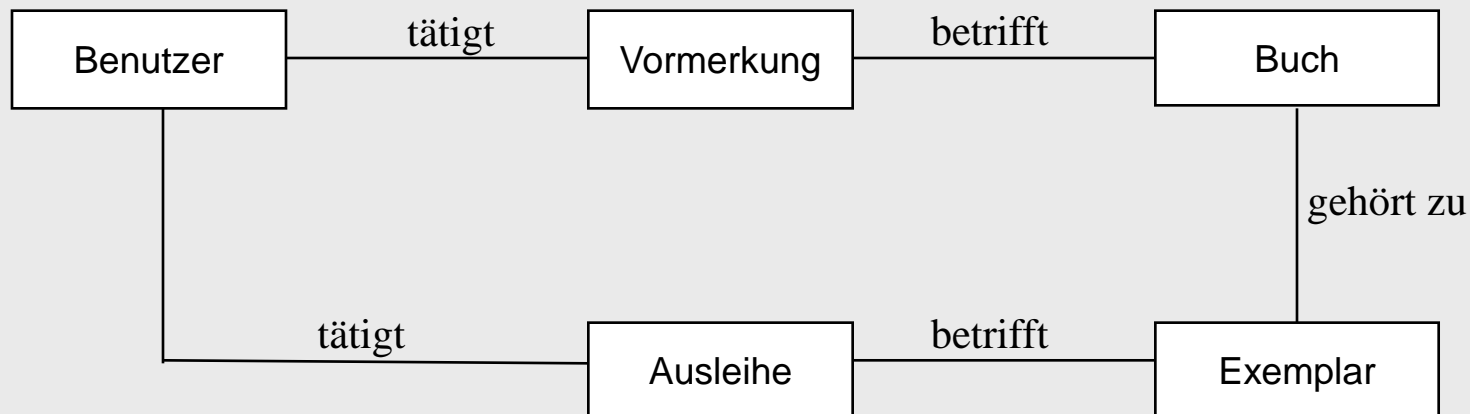
- Parallel zur Klassenidentifikation erfolgt die Ermittlung der Beziehungen zwischen den bereits identifizierten Klassen.
- Kandidaten dafür lassen sich ebenfalls anhand der Anforderungsszenarien herleiten, z.B. im Zusammenhang mit
 - Aktivitäten und
 - Datenabhängigkeiten
(z.B. zwischen Buch und Exemplar: zu einem Buch gehören i.A. mehrere Exemplare, die gemeinsam auf Informationen des zugehörigen Buch-Objekts verweisen, wie etwa Angaben zu Autor und Titel)
- Dabei sind zunächst genauere Angaben, etwa zur Multiplizität bzw. zur Art der betrachteten Assoziation, nicht erforderlich (werden später im OOD ergänzt).

Herleitung von Beziehungen aus Aktivitäten

- Aktivitäten:
Eine aus den Anforderungen zu vermutende Interaktion zwischen zwei Klassenkandidaten deutet bereits auf eine mögliche Assoziation zwischen beiden Klassen hin.
- Beispiel:
 - Der Benutzer tätigt (persönlich) die Ausleihe eines Buchexemplars (Assoziation zwischen Benutzer und Ausleihe).
 - Die Ausleihe betrifft ein Exemplar (Assoziation zwischen Ausleihe und Exemplar).
 - Der Benutzer beantragt die Vormerkung für ein Buch (Assoziation zwischen Benutzer und Vormerkung).
 - Die Vormerkung betrifft ein Buch (Assoziation zwischen Vormerkung und Buch).

Beispiel für Klassendiagramm: Bibliothek

- Für die Klassen im Bibliotheksbeispiel ergeben sich die im folgenden Diagramm dargestellten Beziehungen:

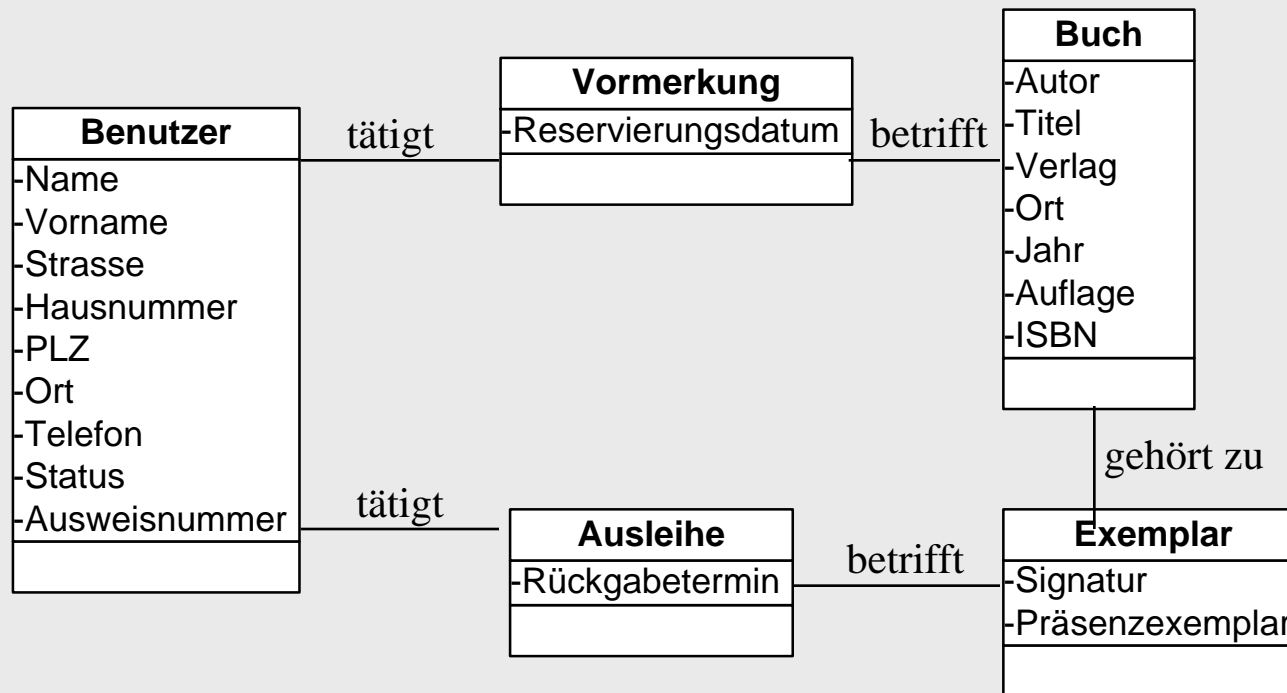


Attribute spezifizieren

- Zu den identifizierten Klassen können zusätzliche Eigenschaften (sog. Attribute) festgelegt werden.
- Soweit zu diesem Zeitpunkt möglich, werden für Attribute die folgenden Eigenschaften spezifiziert:
 - Name des Attributs
 - Beschreibung des Attributs
 - Sichtbarkeit
 - Typ des Attributs
- Beispiel: Bibliothek
 - Zur Klasse Exemplar bietet sich das Attribut "Präsenzexemplar" (vom Typ Boolean) an.

Beispiel für weitere Attribute: Bibliothek

- Die identifizierten Klassen werden über die Anforderung hinaus mit sinnvollen Zusatzinformationen ergänzt. Die Sichtbarkeit wird bei den angegebenen Attributen in aller Regel privat sein.



Übung

Objektorientierte Analyse mittels UML: Waschmaschine

Beschreibung:

Im Folgenden werden im Rahmen einer Produktbeschreibung (Auszug) die Anforderungen an eine durch Software gesteuerte Waschmaschine definiert:

Es soll zwei generelle Betriebsmodi geben: Waschen und Schleudern. Beim Waschen dreht sich die Trommel mit einer Drehzahl von 30 Umdrehungen in der Minute. Beim Schleudern dreht sich die Trommel mit einer Drehzahl von 500 Umdrehungen pro Minute. Die Waschmaschine verfügt über 6 Programme (Feinwäsche, Buntwäsche, Wolle, Normal, Kurzprogramm, Schleudern). Neben den Programmen können folgende Temperaturen des Waschwassers vorgewählt werden: 30°, 60°, 95°. Die Waschmaschine steuert den Wasserzufluss an einem elektrischen Ventil, das mit dem Wasserhahn direkt verbunden ist. Das gewünschte Waschprogramm und die Temperatur werden über eine Bedienkonsole vorgewählt. Im Inneren der Trommel befindet sich ein Temperatursensor. Dessen Messwerte sollen über einen seriellen Datenbus alle 60 Sekunden gelesen und ausgewertet werden.

Um Wasser zu sparen befindet sich noch ein Wassergütesensor in der Trommel. Erreicht die Wassergüte einen unteren Schwellwert, wird weiteres Leitungswasser zugeführt.

Aufgabe:

Es soll die SW für die Steuerung der Waschmaschine modelliert werden. Identifizieren Sie dazu die in Frage kommenden Objekte, klassifizieren Sie diese. Sie zugehörige Attribute und Assoziationen. **Zeit:** 20 Minuten



Zum Schluss dieses Abschnitts ...

Noch Fragen ??

Inhalt

5 Methoden

5.2 Architektur und Design

5.2.1 Software-Entwurf

5.2.2 Objektorientierte Analyse

5.2.2.1 Einführung und Überblick

5.2.2.2 Grundkonzepte und Elemente der UML

5.2.2.3 Statische Modellierung

5.2.2.4 Dynamische Modellierung

5.2.3 Objektorientiertes Design

5.2.4 Entwurfsmuster (Design-Pattern)

5.2.5 Modellbasierte Entwicklung

5.2.6 Architektur von Embedded Echtzeit-Systemen

5.2.7 Standard-Architekturen am Beispiel AUTOSAR

Externe Szenarien erstellen

- Szenarien sind Verfeinerungen der Anwendungsfälle, die in Form von Interaktionsdiagrammen dokumentiert werden. Interaktionen können ihrerseits anhand einer der beiden folgenden, äquivalenten Diagrammarten dargestellt werden:
 - Sequenzdiagramm
 - Kommunikationsdiagramm
- Szenarien tragen im Wesentlichen dazu bei
 - den Fluss der **Botschaften** durch das System zu definieren.
- Im Folgenden Beschränkung auf **externe Szenarien**, also
 - Interaktionen zwischen dem Software-System und seiner Umgebung,
 - ohne weitergehende Implementierungsdetails zu berücksichtigen, die erst im Feinentwurf zum Vorschein kommen werden.

Externe Szenarien erstellen

- Aus jedem Anwendungsfall werden oft mehrere Szenarien abgeleitet:
 - Variationen von Anwendungsfällen führen zu unterschiedlichen Szenarien.
- Standardausführung und Alternativen:
 - **Primäre Szenarien** stellen die fundamentalen Funktionen des Systems dar.
 - **Sekundäre Szenarien** präsentieren Variationen primärer Szenarien. Sie beschreiben Ausnahmesituationen und enthalten die weniger oft verwendeten Funktionen.

Sequenzdiagramm

➤ Objekte

- Die an dem dargestellten Szenario beteiligten Objekte werden durch das übliche UML-Objektsymbol (Rechteck mit Beschriftung) repräsentiert.
- Zu jedem beteiligten Objekt gibt es eine gestrichelte „Lebenslinie“ (Lifeline), die vom Objektsymbol vertikal nach unten verläuft.

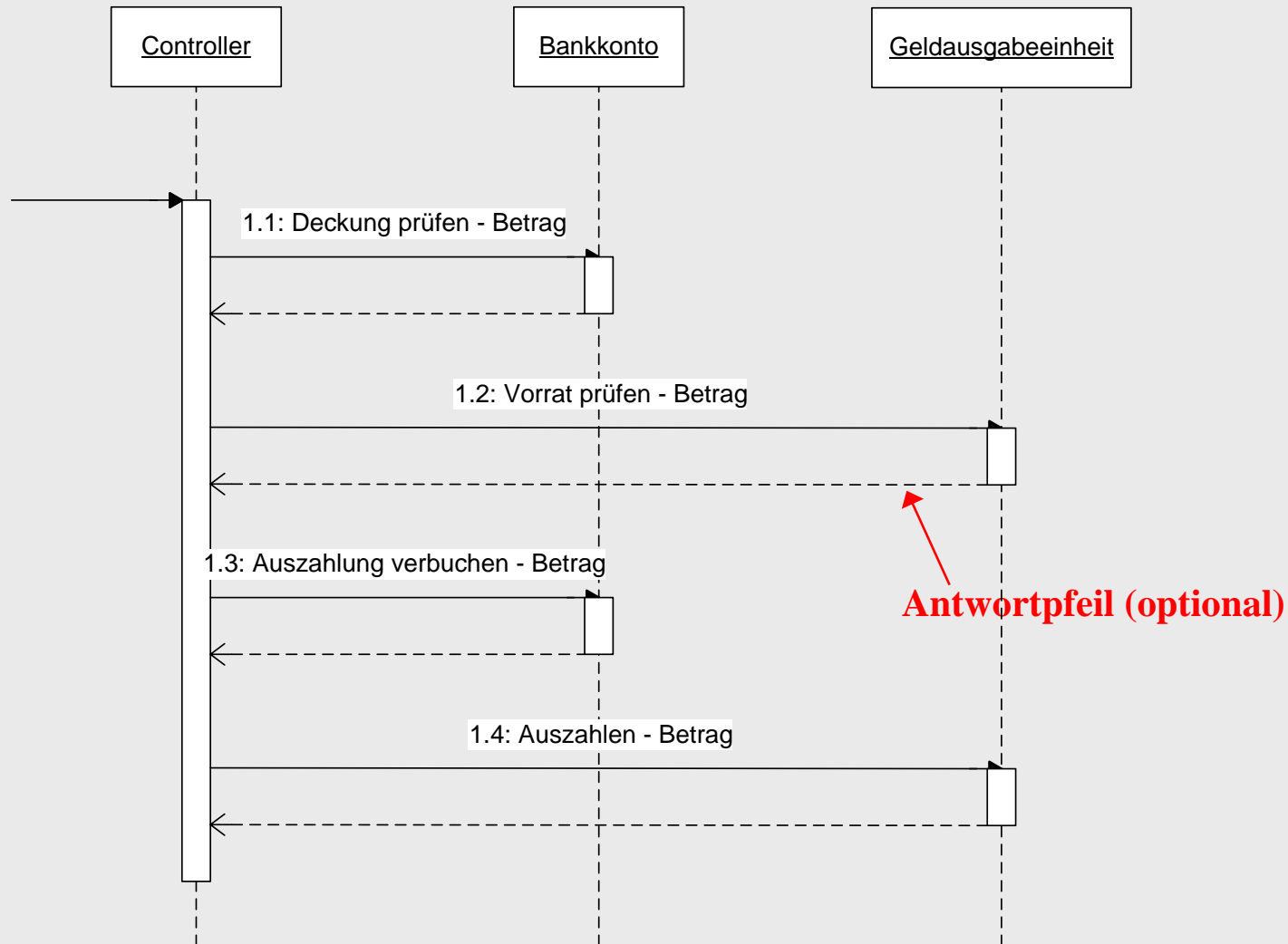
➤ Zeitachse

- Eine virtuelle Zeitachse bestimmt die zeitliche Ordnung der im Sequenzdiagramm dargestellten Ereignisse.
- Sie verläuft parallel zu den Lebenslinien der Objekte, vertikal von oben nach unten.

➤ Nachrichten

- Jede Kommunikation zwischen zwei Objekten wird durch eine Nachricht modelliert.
- Eine Nachricht wird durch eine gerichtete Kante zwischen den Lebenslinien von Sender- und Empfängerobjekt repräsentiert.

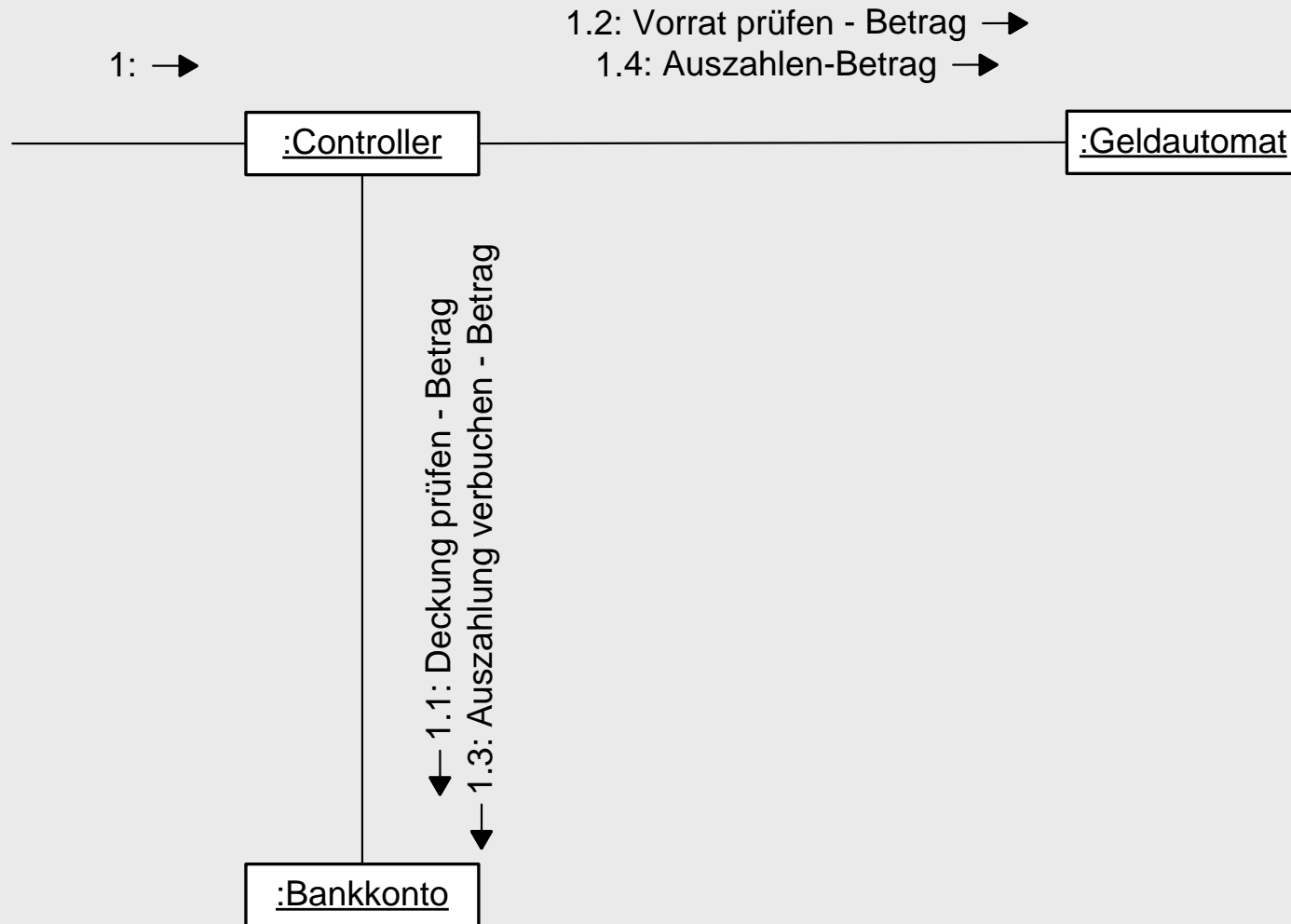
Beispiel für Sequenzdiagramm: Geldautomat



Kommunikationsdiagramm

- Im Diagramm werden die beteiligten **Objekte** dargestellt, ggf. mit Rollenangaben zur Verdeutlichung der Bedeutung des Objektes innerhalb des jeweiligen Szenarios
- Beziehungen zwischen den beteiligten Objekten werden durch **Verknüpfungskanten** dargestellt
- Der Nachrichtenaustausch (Interaktion) zwischen den beteiligten Objekten erfolgt entlang der Verknüpfungskanten
- Die zeitliche Reihenfolge der Nachrichten wird durch ein Nummerierungsschema (z.B. 1, 2, 2.1 2.2, 3, 4) ausgedrückt

Beispiel für Kommunikationsdiagramm: Geldautomat



Kommunikationsdiagramm <-> Sequenzdiagramm

- Das Sequenzdiagramm und das Kommunikationsdiagramm sind zwei semantisch äquivalente Arten von Interaktionsdiagrammen.
- Das Sequenzdiagramm betont den **temporalen** Ablauf und die zeitliche Reihenfolge des Nachrichtenaustauschs.
- Das Kommunikationsdiagramm betont die (**statischen**) Verknüpfungen zwischen den interagierenden Objekten.

Zum Schluss dieses Abschnitts ...

Noch Fragen ??