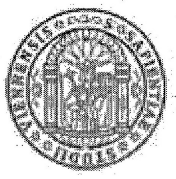


universität
wien

Modellierung

TEIL 1

Modul "Grundlagen der Modellierung" (PA.MOD.MD.VO)

Vorlesung im SS 2011

Univ. Prof. Dr. Wolfgang Klas
Univ. Prof. Dr. Dimitris Karagiannis

Fakultät für Informatik
Universität Wien

universität wien

Hinweise zu den LV-Unterlagen

Die Unterlagen zu dieser Lehrveranstaltung bestehen zum einen aus traditionellen Lehrveranstaltungsunterlagen wie zum Beispiel Folien, zum anderen aus Unterlagen in Form der referenzierten Literatur. Das erfolgreiche Studium des Lehrinhalts dieser Lehrveranstaltung erfordert auch die eingehende Beschäftigung mit der referenzierten Literatur (Bücher, Papers, etc.). Weiters wird dringend der begleitende Besuch der auf die Vorlesung abgestimmten Übung empfohlen.

Acknowledgment

Die Aufbereitung des Lehrinhalts sowie die Durchführung der Lehrveranstaltung und der zugehörigen Übung erfolgt im Team. Dank gebührt daher allen beteiligten Mitarbeitern.

Hinweis zur LV-Prüfung

Prüfungsrelevant ist immer jener Stoff, der im zuletzt durchgeführten Lehrveranstaltungszyklus Gegenstand der Lehrveranstaltung war. Der Umfang des prüfungsrelevanten Stoffs wird jeweils über die im zuletzt durchgeführten Lehrveranstaltungszyklus **schriftlich** verfügbar gemachten LV-Unterlagen definiert.

Inhaltsverzeichnis
TEIL 1

1 Modul "Grundlagen der Modellierung" 1-1

1.1 Einbettung in den Studienkontext ....-1
1.2 Termine der Vorlesung im SS 2011 ....-3
1.3 Unterlagen zur Vorlesung ....-3
1.4 Inhalt der Vorlesung ....-4
1.5 Zielsetzung ....-5

2 Was ist unter Modellierung zu verstehen? 2-1

2.1 Modelle ....-2
2.1.1 Herkunft des Begriffs Modell ....-2
2.1.2 Was ist ein Modell? ....-3
2.1.3 Mentale Modelle ....-5
2.2 Warum Modellierung? ....-6
2.2.1 Modellierungssprachen ....-9
2.2.2 Modell vs. Diagramm ....-12
2.3 Arten von Modellen ....-13
2.3.1 Allgemeine Überlegungen ....-13
2.3.2 Datenorientierte Modellierung ....-14
2.3.3 Prozessorientierte Modellierung ....-17
2.3.4 Objektorientierte Modelle ....-20
2.3.5 Ontologien ....-23
2.3.6 Modelle aus dem Bereich Wissensrepräsentation ....-26

3 Das Entity-Relationship-Modell 3-1

3.1 Grundlagen des Entity-Relationship-Modells ....-2
3.1.1 Eckdaten ....-2
3.1.2 Nutzung des Entity-Relationship Modells ....-4
3.1.3 Grundbegriffe ....-6
3.1.4 Sprachen zur Notation von ER-Modellen ....-8
3.1.5 Abstraktionskonzepte (allgemein) ....-9
3.1.6 Entities und Attribute ....-11
3.1.7 Relationships ....-19
3.1.8 IS-A-Beziehungen ....-28
3.1.9 Beispiel ....-35
3.2 Erweiterungen und Spezialfälle der ER-Modellierung ....-39
3.2.1 Min,Max-Notation ....-39
3.2.2 Modifizierte Chen-Notation (MC-Notation) ....-41
3.2.3 Gegenüberstellung Min-Max-Notation, Chen-Notation und MC-Notation ....-44
3.2.4 Weak Entity-Typ (Schwache Entity) ....-45

3.2.5 Uminterpretierter Beziehungstyp ....-47
3.2.6 Historische Daten ....-49

4 Objektorientierte Modellierung 4-1

4.1 Entwicklung objektorientierter Modellierung ....-2
4.2 Grundlegende Konzepte der objektorientierten Modellierung ....-4
4.3 Umsetzung objektorientierter Modellierung in UML ....-5
4.3.1 Wichtige UML Schemata für statische Aspekte ....-7
4.3.2 Wichtige UML Schemata für Dynamische Aspekte ....-8
4.3.3 Wichtige UML Schemata für softwareorganisatorische Aspekte ....-9
4.3.4 Zusammenspiel wichtiger Schemata in UML ....-10
4.3.5 Modelle, Diagramme und Werkzeuge ....-11
4.3.6 UML 2.0 Diagrammtypen ....-12

5 Anwendungsfalldiagramm 5-1

5.1 Überblick Use Case / Anwendungsfall ....-2
5.2 Definitionen ....-10
5.3 Akteure, Anwendungsfälle und Kommunikationsbeziehungen ....-11
5.4 Beziehungen zwischen Anwendungsfällen ....-15
5.5 Beziehungen zwischen Akteuren ....-22
5.6 Beschreibung eines Anwendungsfalls ....-24
5.7 Identifizierung von Anwendungsfällen ....-27
5.8 Die wichtigsten Regeln zur Anwendungsfallmodellierung ....-28

6 Klassen- und Objektdiagramme 6-1

6.1 Klassen- bzw. Objektdiagramme ....-2
6.1.1 Klassen ....-3
6.1.2 Objekt ....-8
6.1.3 Allgemeine Assoziation ....-9
6.1.4 Qualifizierte Assoziation ....-11
6.1.5 Attributierte Assoziation (Assoziationsklasse) ....-12
6.1.6 Mehrgliedrige Assoziation ....-14
6.1.7 Aggregation ....-15
6.1.8 Komposition ....-16
6.1.9 Generalisierung ....-17

7 Aktivitätsdiagramm 7-1

8 Zustandsdiagramm 8-1

8.1 Allgemeines ....-2
8.2 Darstellung von Zustandsdiagrammen ....-4
8.3 Ereignisse ....-11
8.4 Verfeinerung ....-12
8.5 Komplexer Übergang ....-17
8.6 Segmentierter Übergang ....-18
8.7 Gesamtbeispiel Fernseher ....-19

8.8 Exkurs: Zustandsdiagramme vs. Aktivitätsdiagramme .....-20

9 Sequenzdiagramm 9-1

9.1 Exkurs Interaktionsdiagramme .....-3
9.2 Übersicht Sequenzdiagramm .....-7
9.3 Digrammrahmen .....-9
9.4 Lebenslinie .....-10
9.4.1 Ereignisspezifikation .....-11
9.4.2 Reihenfolge .....-12
9.4.3 Ausführungsspezifikation .....-13
9.4.4 Aktives Objekt .....-14
9.5 Nachricht .....-15
9.5.1 Spezielle Nachrichtenarten .....-16
9.5.2 Beispiel .....-17
9.5.3 Zeiteinschränkungen .....-18
9.5.4 Beispiel .....-19
9.6 Zustandsinvariante .....-20
9.7 Kombinierte Fragmente .....-21
9.7.1 Notation .....-22
9.7.2 Operatorarten .....-23
9.7.3 Verzweigungen und Schleifen: alt-Operator .....-24
9.7.4 Verzweigungen und Schleifen: loop-Operator .....-25
9.7.5 Verzweigungen und Schleifen: opt- und break-Operator .....-26
9.7.6 Parallelität und Ordnung: seq-Operator / strict-Operator .....-27
9.7.7 Parallelität und Ordnung: par-Operator .....-28
9.7.8 Parallelität und Ordnung: critical-Operator .....-29
9.7.9 Filterungen und Zusicherungen: ignore-Operator / consider-Operator .....-30
9.7.10 Filterungen und Zusicherungen: assert-Operator / neg-Operator ..-31
9.8 Modularisierung .....-32
9.8.1 Interaktionsreferenz .....-32
9.8.2 Fortsetzungsmarke .....-34
9.8.3 Verknüpfungspunkt .....-35
9.9 Exkurs: Sequenzdiagramm vs. Aktivitätsdiagramm .....-37

1 Modul “Grundlagen der Modellierung”

1.1 Einbettung in den Studienkontext

- Modul im Bachelorstudium Informatik und Bachelorstudium Wirtschaftsinformatik:

  - Abhaltung im Sommersemester:

050039 1	VO	2h	Modellierung
050008 *	UE	2h	Modellierung

- Vorlesungen behandeln Stoff vorwiegend aus konzeptioneller Sicht

- Übungen vertiefen den Stoff und erlauben praktischerer Durchdringung der Fragestellungen (Übung und Vorlesung stehen in Zusammenhang)

- Die Übung besitzt einen immanenten Prüfungscharakter. Hier wird die Leistung der Studenten und Studentinnen Woche für Woche beurteilt.

- Die Prüfung zur Vorlesung erfolgt im Rahmen des Bachelorstudiums als Lehrveranstaltungsprüfung. Der erste mögliche Termin findet am Ende des Semesters statt.

- Um zu einer Lehrveranstaltungsprüfung antreten zu können ist zuvor unbedingt eine Anmeldung im ISWI erforderlich! Die Anmeldefrist ist dabei ca. 3 Wochen bis eine Woche vor dem jeweiligen Prüfungstermin. Die Abmeldefrist endet i.d.R. 2 Tage vor dem Prüfungstermin.

  Details siehe http://cs.univie.ac.at/students/anmeldung/

- Studierende des 033 526 Bakkalaureatsstudiums Wirtschaftsinformatik müssen den Modul PA.MOD als Ersatz für WI/MTM absolvieren (siehe Äquivalenzverordnung).

- Weitere Infos zu Vorlesung und Modul im Online-Lehrveranstaltungsverzeichnis der Universität Wien

- Weitere Infos zum Studienplan: siehe Web-Site der Fakultät

1.2 Termine der Vorlesung im SS 2011

- VO Termine: siehe Online-Lehrveranstaltungsverzeichnis

- Lehrveranstaltungsprüfung zur Vorlesung:
  - Ende Semester, Termin wird bekanntgegeben.
    Rechtzeitige Anmeldung im ISWI erforderlich. Fristen beachten!!
  - Weitere Prüfungstermine werden im ISWI bekanntgemacht.

1.3 Unterlagen zur Vorlesung

- Diese Arbeitsunterlagen dienen zur Unterstützung der Vorlesung. Sie ersetzen NICHT die relevante Literatur. Die Arbeitsunterlagen werden im Verlauf der Vorlesung zur Verfügung gestellt.

- Ein erfolgreicher Abschluss der Lehrveranstaltung erfordert i.d.R eine eingehende Beschäftigung mit der relevanten Literatur!

- Relevante Literatur wird in den Kapiteln angegeben oder in der Vorlesung genannt.

1.4 Inhalt der Vorlesung

In der Vorlesung Modellierung sollen die grundlegenden Modellierungstechniken und -methoden vermittelt werden.

Folgende Schwerpunkte werden behandelt:

- Was ist unter Modellierung zu verstehen?

- Entity-Relationship-Modell

- Objektorientierte Modellierung

- Prozessmodelle

- Meta Modellierung

- Umsetzung einer Modellierungsmethode

- Einblick in Standards

1.5 Zielsetzung

- Einer der zentralen Bereiche in der Informatik und ihren Anwendungen (siehe auch Ausprägungsfächer des Studiums) stellt die geeignete Modellierung von Informations- und Kommunikationssystemen dar.

- Daher widmet sich diese Vorlesung den Grundlagen der Modellierungstechniken und -methoden, unabhängig vom Anwendungsfeld.

- Diese Modellierungstechniken und -methoden sind Grundlage für die Planung und Realisierung von IT-Systemen.

- Spezielle oder weiterführende Methoden werden gegebenenfalls im weiteren Verlauf des Studiums oder im konkreten Kontext eines Anwendungsfachs vermittelt.

2 Was ist unter Modellierung zu verstehen?

Inhalt

2.1 Modelle ..... -2
2.1.1 Herkunft des Begriffs Modell ..... -2
2.1.2 Was ist ein Modell? ..... -3
2.1.3 Mentale Modelle ..... -5
2.2 Warum Modellierung? ..... -6
2.2.1 Modellierungssprachen ..... -9
2.2.2 Modell vs. Diagramm ..... -12
2.3 Arten von Modellen ..... -13
2.3.1 Allgemeine Überlegungen ..... -13
2.3.2 Datenorientierte Modellierung ..... -14
2.3.3 Prozessorientierte Modellierung ..... -17
2.3.4 Objektorientierte Modelle ..... -20
2.3.5 Ontologien ..... -23
2.3.6 Modelle aus dem Bereich Wissensrepräsentation ..... -26

2.1 Modelle

2.1.1 Herkunft des Begriffs Modell

Das Wort "Modell" entstand aus dem italienischen "modello" (ursprünglich hervorgegangen aus "modulo", einem Maßstab in der Architektur) und gehörte bis ins 18. Jahrhundert der Fachsprache der bildenden Künstler an. Um 1800 verdrängte Modell im Deutschen das ältere, direkt vom lateinischen "modulus" entlehnte Wort "Model" (Muster, Form, z. B. Kuchenform), das noch im Verb "ummodeln" und einigen Fachsprachen fortlebt.

(siehe z.B.: http://de.wikipedia.org/wiki/Modell_(Begriff))

2.1.2 Was ist ein Modell?

- Breit akzeptierte allgemeine Modelltheorie 1973 von Herbert Stachowiak vorgeschlagen:
  - Modellbegriff nicht auf eine Fachdisziplin festgelegt, sondern domänenübergreifend, also allgemein anwendbar.

  - 1. Merkmal: Abbildung
    Modell ist immer ein Abbild von etwas, ein Artefakt, eine Repräsentation natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können.

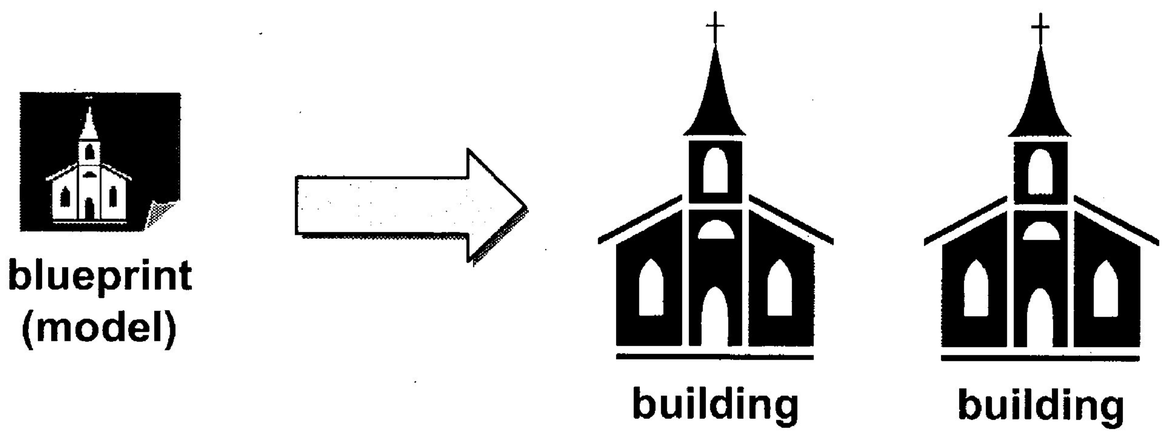
  - 2. Merkmal: Verkürzung
    Ein Modell erfasst nicht alle Attribute des Originals, sondern nur diejenigen, die im Kontext des Modells relevant erscheinen.

  - 3. Merkmal: Pragmatismus
    Pragmatismus = Orientierung am Nützlichen. Für wen? Warum? Wozu? Modell wird innerhalb einer bestimmten Zeitspanne und zu einem bestimmten Zweck eingesetzt. Es erfolgt somit immer eine Interpretation des Modells.

- Modell ist daher Abstraktion: die bewusste Vernachlässigung bestimmter Merkmale, um die für den Modellierer oder den Modellierungszweck wesentlichen Modelleigenschaften hervorzuheben.

- Allgemein: bestimmter Abstraktionsgrad wird nicht vorausgesetzt.
  Selbst einfachste Abbildungen der Realität (z.B. das Messen der Körpergröße eines Menschen) sind nach dieser allgemeingültigen Definition bereits ein Modell.

- In der Regel jedoch (präzise) Vorgabe an etwas, das Geschaffen werden muß



- Informatik: Modelle oft sehr präzise und mathematisch formulierbar.
  Theorie der Formalen Sprachen ist eigenständiges Wissensgebiet in der Theoretischen Informatik.

2.1.3 Mentale Modelle

- Mentale Modelle sind
  - Annahmen
  - Generalisierungen
  - Bilder
  - Eindrücke

  die beeinflussen, wie wir die Welt sehen und dementsprechend reagieren

- Wir alle haben z.B. Modelle der Geschäftswelt in der wir arbeiten und der Geschäftabläufe, die dort passieren

- Mentale Modelle beeinflussen unsere Reaktionen auf Ereignisse in dieser Welt

- Mentale Modelle der einzelnen Beteiligten stimmen nicht immer überein

- Dies bezeichnen wir als „Missverständnis“, Missverständnisse führen zu schlecht funktionierenden Systemen

- Es ist wichtig, ein von allen getragenes mentales Modell zu entwickeln

2.2 Warum Modellierung?

- Modellierung = Vorgehensweise zur Erstellung eines Modells

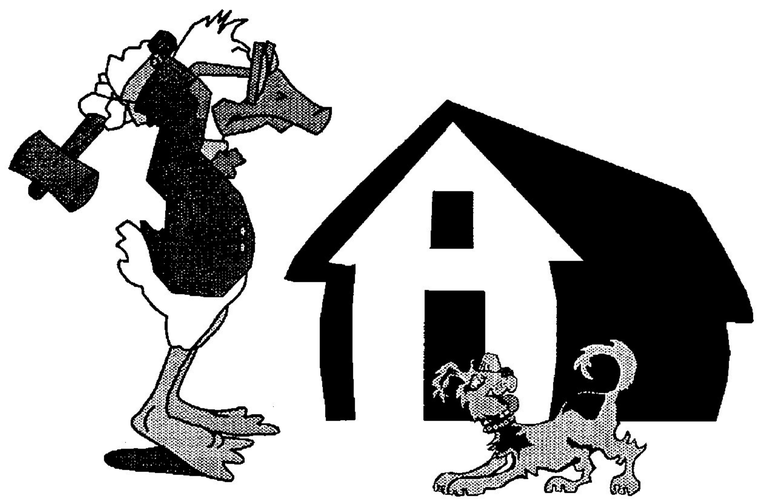
- Ein Modell ist die “Vereinfachung der Realität”

- Wir erstellen Modelle, damit wir die Realität besser verstehen können

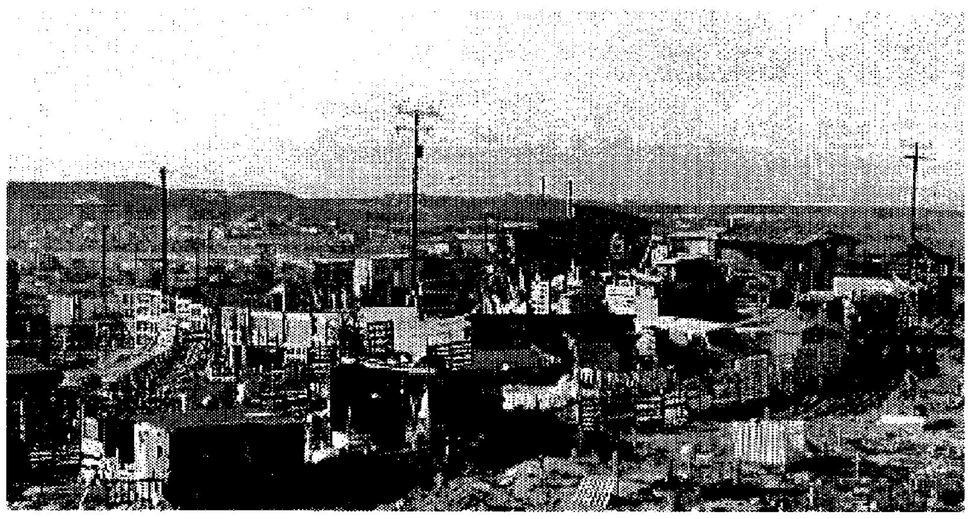
- Wir bilden Modelle von komplexen Systemen,
  - da wir sie nicht in ihrer Gesamtheit verstehen können,
  - um unterschiedlichen Stakeholders unterschiedliche Sichten auf ein System zu gewähren,
  - um einen gewissen Aspekt eines Systems zu einem gewissen Zeitpunkt zu betrachten,
  - um Abschätzungen von Aufwand/Kosten/Auswirkungen in komplexen, zeit- und kostenkritischen Systemen zu ermöglichen
  - um Handlungsanweisungen, Instruktionen zum Bau komplexer Systeme ableiten zu können.

Analogie Hausbau (auch ein komplexes System)

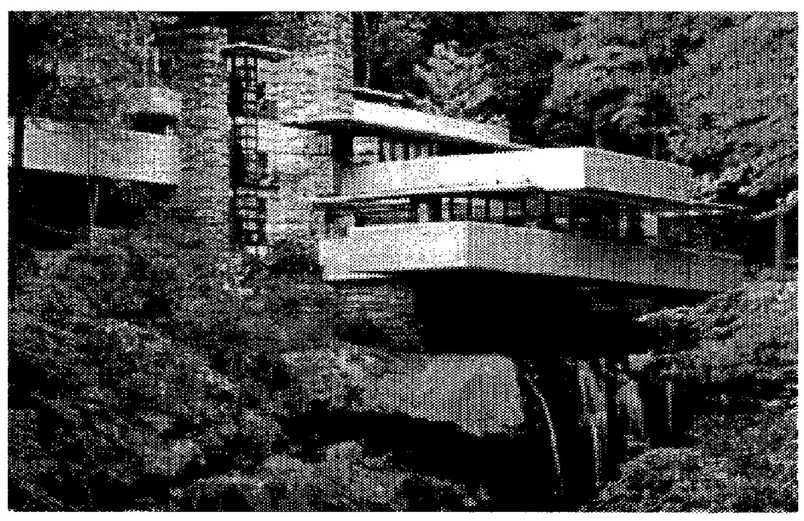
Nur Hundehütten werden ohne einen Bauplan erstellt



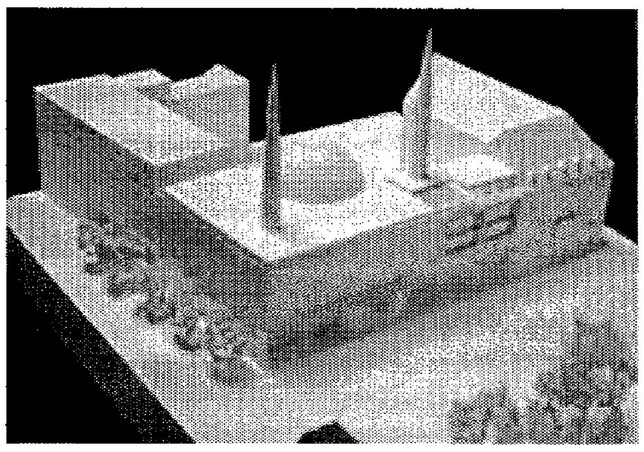
- Die Aufgabe:

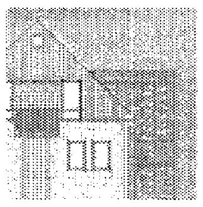


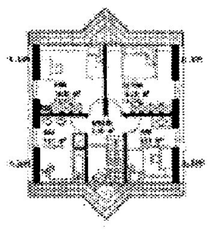
- Die Vision

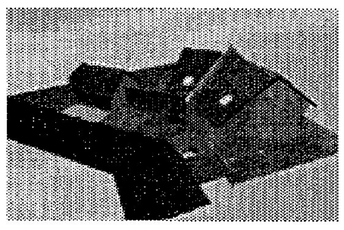


- Die Modelle









2.2.1 Modellierungssprachen

- Wir müssen uns auf ein gemeinsames Mentales Modell verständigen
- Mentale Modelle jedes einzelnen müssen kommuniziert und hinterfragt werden, um die Relevanz zu verstehen
- Kommunikation muß auf einer gemeinsamen Sprache basieren
- Eine Modellierungssprache hilft mentale Modelle in einer Syntax auszudrücken

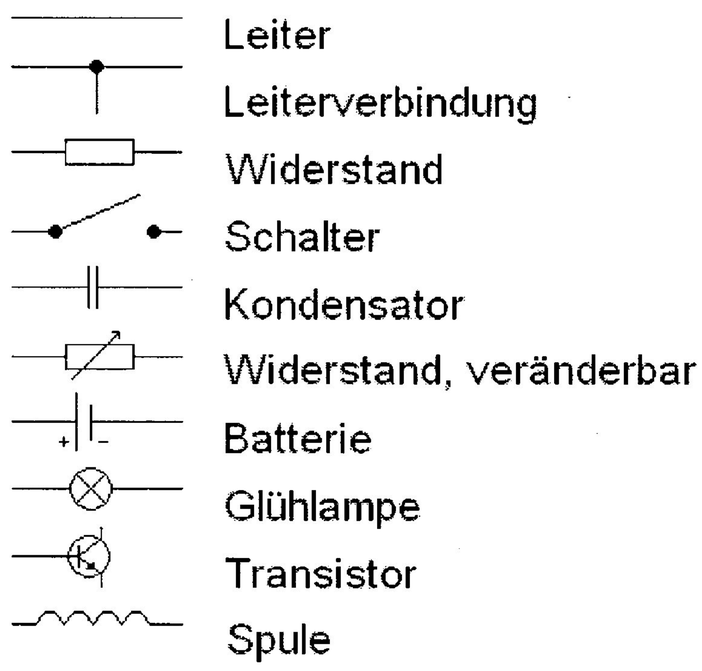
- Modellierungssprachen bedienen sich sehr oft einer grafischen Notation.

- Modellierungssprachen dienen zur
  - Visualisierung
  - Spezifikation
  - Gestaltung
  - Dokumentation

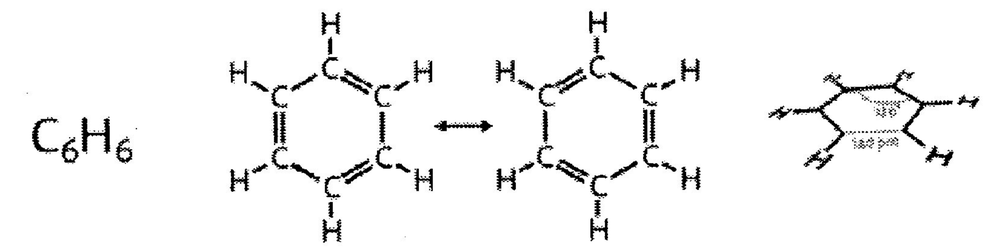
  der Artefakte eines Systems

- Sprache = Syntax + Semantik
  - Syntax = Regeln mit Hilfe derer Elemente einer Sprache (z.B. Wörter) in reguläre Ausdrücke (z.B. Sätze) zusammengefaßt werden
  - Semantik = Regeln, welche syntaktischen Ausdrücken eine Bedeutung verleihen

- Beispiele für Modellierungssprachen
  - Elektrotechnik:
    grafische Notation von Schaltungen



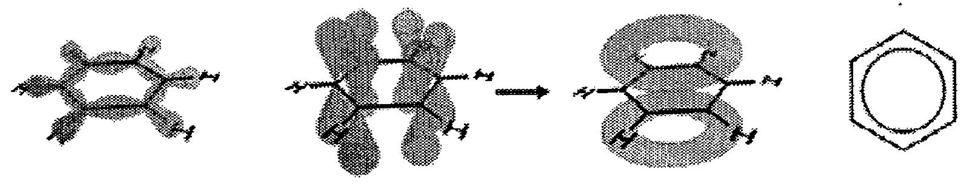
- Chemie:
  grafische Notation durch Strukturformeln:



Benzol
Summenformel

Benzol
Kekulé-Strukturformel
(Mesomerie)

Benzol
planares Hexagon
Kantenlänge 140 pm



Benzol
σ-Bindungen mit hybridisierten sp^2-Orbitalen

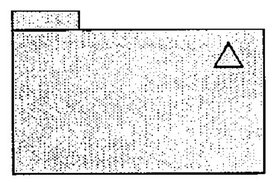
Benzol
6 p_z-Orbitale

Benzol
delokalisierte
π-Orbitalwolke

Benzolring,
vereinfachte
Darstellung

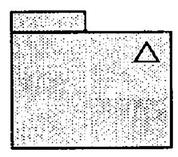
2.2.2 Modell vs. Diagramm

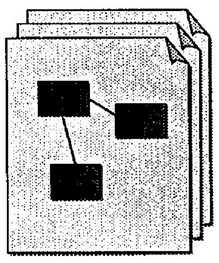
- Ein Modell ist die Abstraktion eines Systems.



- Ein Modell beschreibt das System von einem bestimmten Blickwinkel mit einem bestimmten Abstraktionsniveau und Detaillierungsgrad

- Diagramme
  entsprechen der Dokumentation eines Modells





2.3 Arten von Modellen

2.3.1 Allgemeine Überlegungen

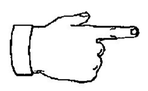
Zentrale Frage: Welches Modell braucht man für welchen Zweck?

- Definiere den Zweck jedes Modells
- Ein Modell muß gemäß seiner Sichtweise und seines Abstraktionsniveaus ein für sich komplettes Bild des Systems darstellen
- Behalte den Zweck des Modells im Auge und vernachlässige - je nach Abstraktionsniveau - Nebensächlichkeiten
- Folge: Unterschiedliche Modelle für unterschiedliche Zwecke!

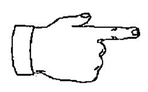
2.3.2 Datenorientierte Modellierung

- ausgehend von statischen Strukturen
  - z.B. Bäume, Listen, Records
    siehe z.B. Datenmodellierung in Programmiersprachen
  - z.B. organisatorischen Gegebenheiten
    siehe z.B. Organigramme
  - ...
- Daten werden (zunächst) funktionsunabhängig betrachtet
- Erstellung eines (globalen) konzeptionellen Datenmodells
  - hilft Redundanz zu vermeiden,
  - erleichtert Integritätssicherung,
  - erhöht Systemstabilität

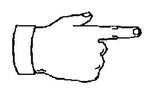
- Datenorientierte Vorgehensweise kann in folgende Schritte eingeteilt werden
  - Relevanter Problemausschnitt (z. B. Einkauf, Buchhaltung) wird definiert, um relevante Daten des Problemausschnitts bestimmen zu können.
  - Logisches Datenmodell wird definiert (Fachkonzept), indem die Daten strukturiert werden. Die Darstellung erfolgt **beispielsweise** durch ein **Entity-Relationship-Modell**

     siehe Abschnitt "3 Das Entity-Relationship-Modell".

  - DV-Konzept: Das logische Datenmodell wird z.B. in ein Datenbankmodell umgesetzt, indem die Begriffswelt des Fachkonzeptes auf den DV-Bereich übertragen wird.

     siehe weiterführende Vorlesungen.

  - Das Datenbankmodell wird in die Sprache eines Datenbanksystems umgesetzt. Erst hier erfolgt eine physische Realisierung unter Berücksichtigung der vorhandenen Möglichkeiten. Häufig wird hierbei SQL eingesetzt.

     siehe weiterführende Vorlesungen.

- verschiedene Techniken wurden entwickelt, besonders zu erwähnen sind:
  - Entity-Relationship Model (ER)
    - prominentester Vertreter auf konzeptioneller Ebene
  - IDEF1X / IDEF - ICAM Definition Languages Family
    - von U.S. Air Force entwickelt
    - basiert auf ER Diagrammen
  - Object Role Modeling (ORM) - Mitte 70er Jahre
    - unterstützt/genutzt von verschiedenen Tools
    - z.B. auch Microsoft Visio for Enterprise Architects
  - Relational Model (RM) - 1970
    - prominentester Vertreter auf der Ebene von Datenbankmodellen
    - wichtigstes Modell zur Organisation von Daten
  - Relational Model/Tasmania (RM/T) - 1979
    - Erweiterungen von RM, keine kommerzielle Bedeutung
  - Barker's Notation - 1990
    - basiert auf ER-Diagrammen
    - z.B. Oracle CASE modelling tools
  - Extended Backus–Naur Form (EBNF Grammars) - 1977
    - ISO-14977 Standard
    - W3C verwendet eine EBNF zur Spezifikation von XML

2.3.3 Prozessorientierte Modellierung

- ausgehend von dynamischen Abläufen
  - z.B. Sequenz, Parallelität, Wiederholungen, Entscheidung, ...
    siehe z.B. Programmfluss in Programmiersprachen
  - z.B. Business Processes
    siehe z.B. Bestellung, Lieferung, Produktion, ...
  - ...

- Abläufe werden (zunächst) datenunabhängig betrachtet

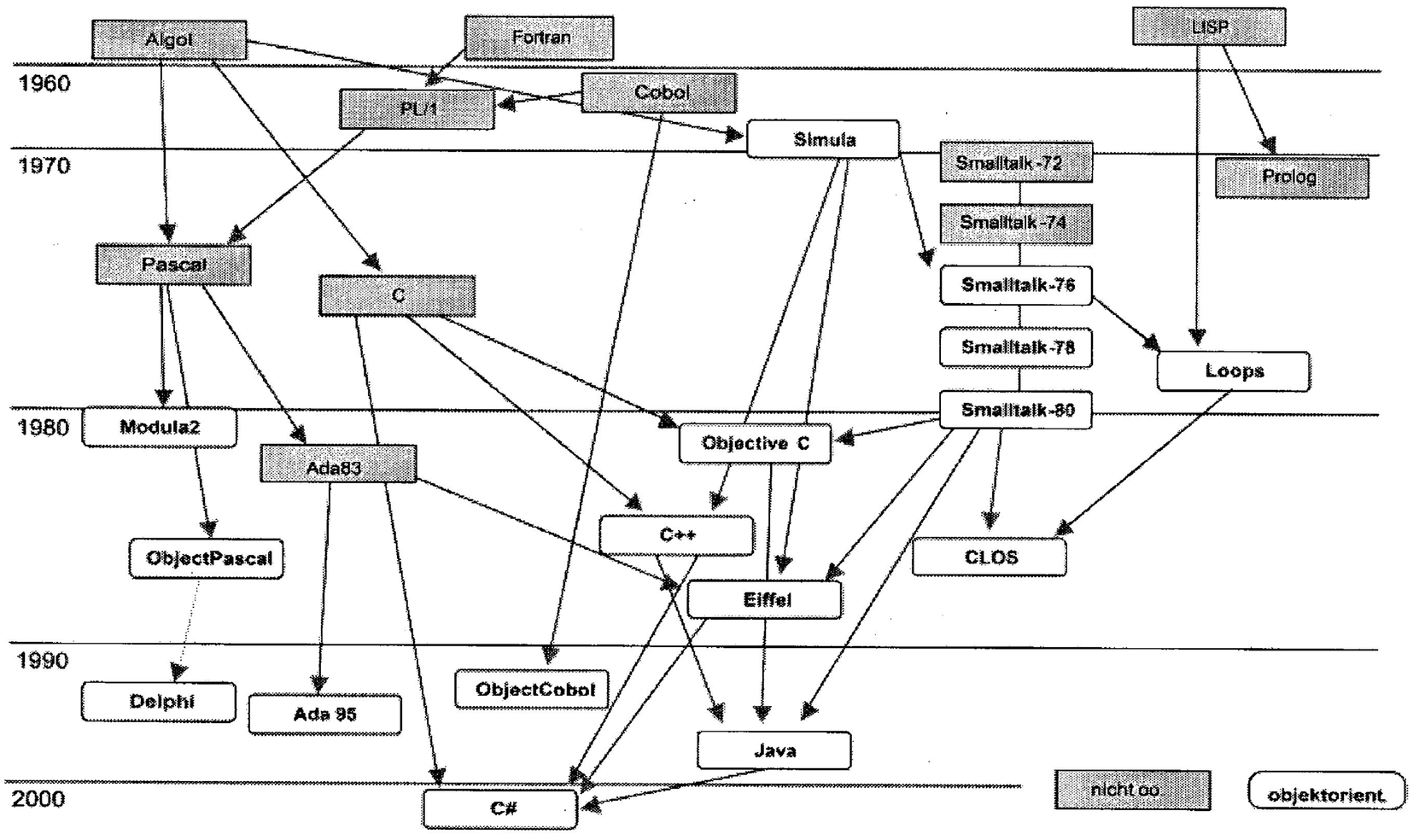
- Erstellung eines (globalen) konzeptionellen Ablaufmodells
  - hilft Redundanz zu vermeiden,
  - erleichtert Integritätssicherung,
  - erhöht Systemstabilität

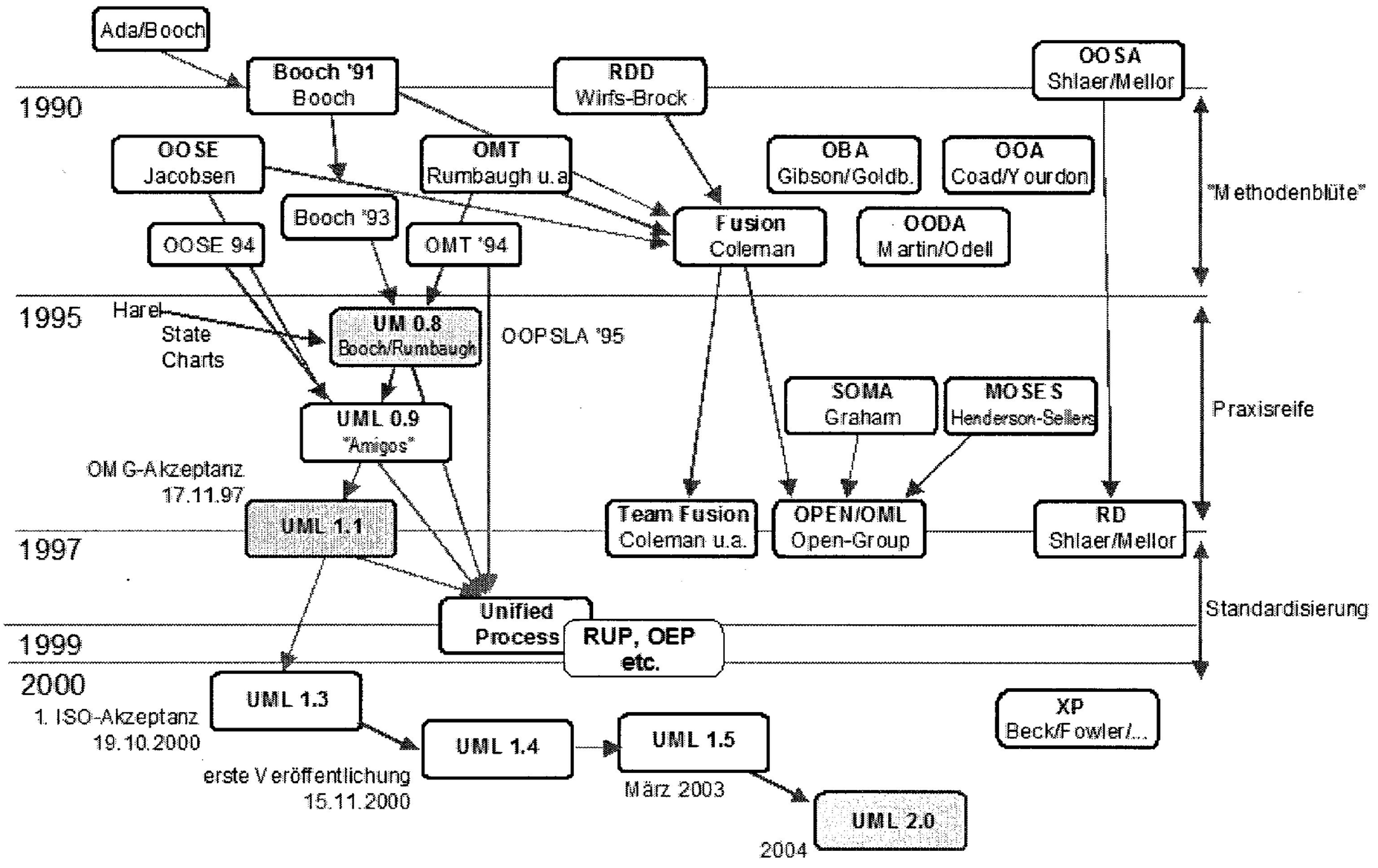
- verschiedene Techniken wurden entwickelt, besonders zu erwähnen sind:
  - verschiedene Diagramme zur Geschäftsprozessmodellierung
  - Aktivitätsdiagramme
    - graphische Darstellung eines Netzes von elementaren Aktionen, die mit Kontroll- und Datenflüssen verbunden sind
    - eine der dreizehn Diagrammarten in der Unified Modeling Language (UML)
  - Anwendungsfalldiagramm
    - graphische Darstellung von Anwendungsfällen und Akteuren mit ihren Abhängigkeiten und Beziehungen
    - eine der dreizehn Diagrammarten in der Unified Modeling Language (UML)
  - Zustandsdiagramme, Statechart - 1984
    - eine grafische Darstellungsform von endlichen Automaten
    - eine der dreizehn Diagrammarten in der Unified Modeling Language (UML)
  - Ereignisgesteuerte Prozesskette (EPK) - 1992
    - semiformalen Beschreibung von Geschäftsprozessen
  - Business Process Modeling Notation (BPMN) - 2002
    - Standardisiert von OMG seit 2006
    - BPMN ist eine graphische Notation für die einzelnen Schritte in einem Geschäftsprozess.
    - prominenter Vertreter auf konzeptioneller Ebene von Geschäftsprozessen

- Business Process Execution Language for Web Services (BPEL4WS) - 2003
  - Prozessbeschreibungssprache zur direkten Ausführung des Prozesses
  - Entwickelt von IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems
- Web Services Business Process Execution Language (WS-BPEL) - 2007
  - Nachfolger von BPEL4WS
  - OASIS und breite Industrieunterstützung

2.3.4 Objektorientierte Modelle

Die Entwicklung objektorientierter Modelle hat ihren Ausgangspunkt im Bereich der Programmiersparchen und wurde übernommen in den Bereich der OO Modellierung.





- OO Modellierung verknüpft (statische) Datenmodellierung mit (dynamischer) Verhaltensmodellierung und bietet Instrumente zur Verknüpfung der beiden Modellierungsansätze.

- Prominentestes Beispiel: Unified Modeling Language (UML)
  - greift verschiedene Entwicklungen/Modelle auf
  - vereinigt und integriert verschiedene Ansätze zu einem homogenen, reichhaltigen Modell

- Die Entwicklung eines Standards in diesem Bereich ist vergleichbar mit der Entwicklung von standardisierten Modellen in anderen Wissenschaftsdisziplinen, zum Beispiel:
  - Elektrotechnik: grafische Notation durch Schaltungsdiagrammen
  - Chemie: grafische Notation durch Strukturformeln

UML als Standard ist daher einer der wesentlichsten Entwicklunsgschritte im Bereich der Entwicklung von Modellsprachen in der Informatik!

2.3.5 Ontologien

- Begriff kommt aus der Philosophie
  - Lehre von den Ordnungs-, Begriffs-, und Wesensbestimmungen des Seienden
  - griechisch: ontos (sein), logos (Wort)

- Verwendung in der Informatik als (einfaches) **Modell einer Wissensdomäne**

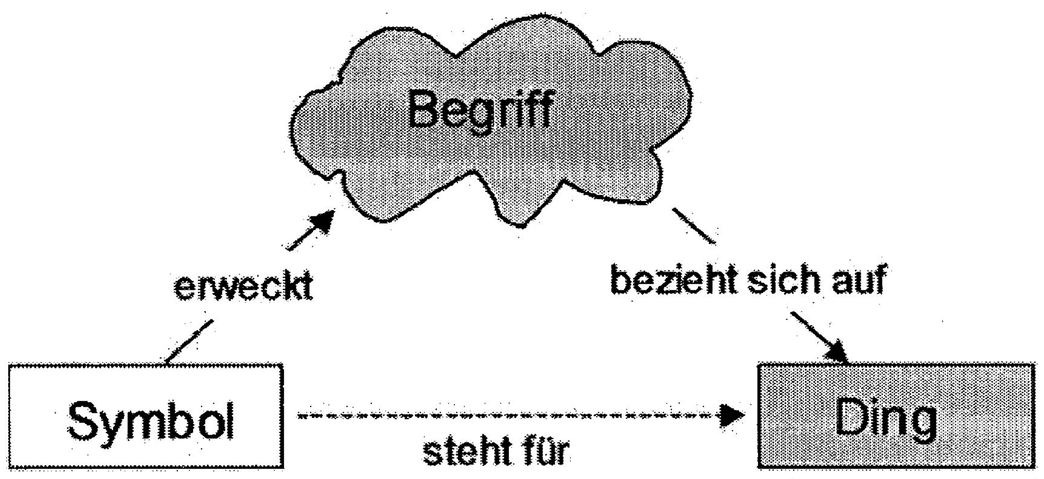
- Verschiedenen Definitionen
  - *„Eine Ontologie ist die formale, explizite Spezifikation einer gemeinsamen Konzeptualisierung."* (T. R. Gruber)
  - *„Ontology is the term used to refer to the shared understanding of some domain of interest [...]"* (M. Unschold)

Ziele und Aufgaben von Ontologien

- Austausch und Teilen von Wissen soll erleichtert werden

- Verbesserung der Kommunikation zwischen Menschen und Softwareagenten (Mensch-Agent; Agent-Agent)

- Beheben von terminologischen Unklarheiten und Aufbau eines gemeinsamen Verständnis



Semiotisches Dreieck

Quelle: Mädche, Staab, Struder, 2001

- Ontologien sollen gewährleisten, dass die Kommunikationspartner unter Begriffen aus einer Domäne dasselbe verstehen

- Ziel ist im Sinne des Semiotischen Dreieck
  - dieselben Symbole bzw. den daraus gebildeten Begriff (z. B. „Jaguar“) für
  - dasselbe Ding (z.B. Automarke, nicht Raubkatze) zu verwenden

- Zu diesem Zweck werden aus einer Domäne
  - Begriffe
  - eine Begriffshierarchie und die
  - Beziehungen zwischen den Begriffen definiert.

- Bedeutung von Ontologien für das Semantic Web
  - Mehrdeutigkeit von Begriffen wird durch die Domänenzugehörigkeit aufgelöst (z.B. „Jaguar“ ist „Automarke“ oder „Raubkatze“)
  - Kommunikation zwischen Softwareagenten wird erst durch Ontologien möglich
  - Suchmaschinen können wesentlich genauere Resultate liefern (Anteil der relevanten Dokumente am Suchergebnis wird erhöht)

- **Ontologien sind eine zentrale Idee für semantischen Datenaustausch**

2.3.6 Modelle aus dem Bereich Wissensrepräsentation

- Wissen über einen Anwendungsbereich muß adäquat modelliert werden.

- Definition geeigneter Konzepte (Terme) zur semantischen Beschreibung von Inhalten

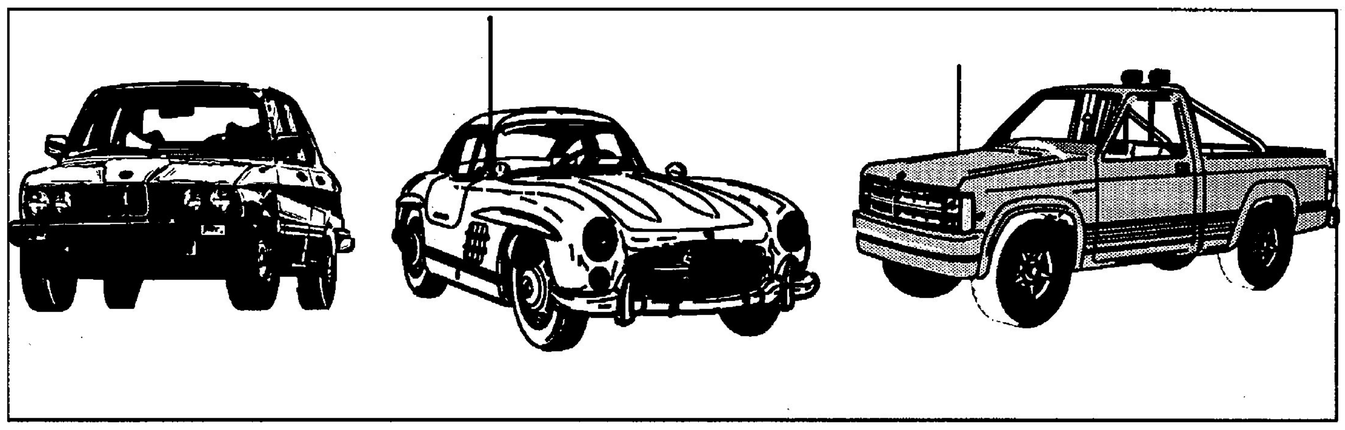
- Zur Modellierung sind verschiedene Techniken auch aus der KI geeignet. Untersucht wurden z.B.

  - semantische Netzwerke:
    läßt sich gut mit OO Datenmodellen modellieren

  - probabilistische Netzwerke:
    wie semantische Netzwerke, nur daß die Kanten Wahrscheinlichkeiten behaftet sind

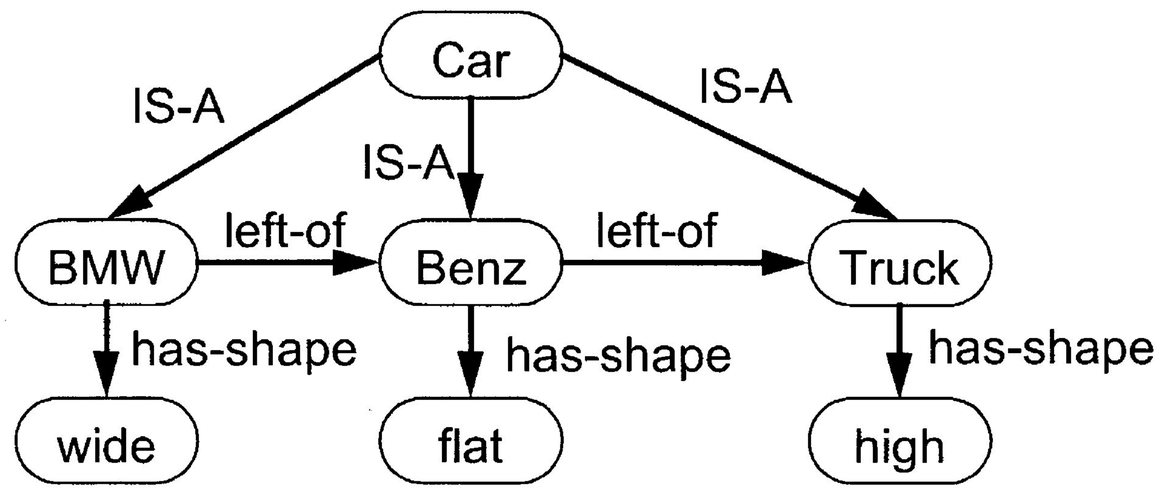
  - logische Sprachen (z.B. Prolog)

Beispiel:



- (möglicherweise) relevantes Wissen über das Bild:
  - Generalisierung: BMW IS-A car
  - räumliche Beziehungen: Benz LEFT-OF Truck
  - Formen: Benz HAS-SHAPE flat

- Kann z.B. in einem semantischen Netzwerk repräsentiert werden



3 Das Entity-Relationship-Modell

Inhalt

3.1 Grundlagen des Entity-Relationship-Modells ..... -2
3.1.1 Eckdaten ..... -2
3.1.2 Nutzung des Entity-Relationship Modells ..... -4
3.1.3 Grundbegriffe ..... -6
3.1.4 Sprachen zur Notation von ER-Modellen ..... -8
3.1.5 Abstraktionskonzepte (allgemein) ..... -9
3.1.6 Entities und Attribute ..... -11
3.1.7 Relationships ..... -19
3.1.8 IS-A-Beziehungen ..... -27
3.1.9 Beispiel ..... -34
3.2 Erweiterungen und Spezialfälle der ER-Modellierung ..... -38
3.2.1 Min,Max-Notation ..... -38
3.2.2 Modifizierte Chen-Notation (MC-Notation) ..... -40
3.2.3 Gegenüberstellung Min-Max-Notation, Chen-Notation und MC-Notation ..... -43
3.2.4 Weak Entity-Typ (Schwache Entity) ..... -44
3.2.5 Uminterpretierter Beziehungstyp ..... -46
3.2.6 Historische Daten ..... -48

3.1 Grundlagen des Entity-Relationship-Modells

3.1.1 Eckdaten

- Entity-Relationship-Modell abgekürzt: ER-Modell oder ERM
- 1976 von Peter Chen in seiner Veröffentlichung *The Entity-Relationship Model* vorgestellt:

  Peter Pin-Shan Chen: The Entity-Relationship Model - Toward a Unified View of Data. In: ACM Transactions on Database Systems, 1/1/1976, ACM-Press, ISSN 0362-5915, S. 9-36
  Download über Uni Wien Rechner/VPN: http://doi.acm.org/10.1145/320434.320440

- Beschreibungsmittel für Generalisierung und Aggregation wurden 1977 von Smith and Smith eingeführt.

  J.M. Smith, D.C.P. Smith: Database Abstractions: Aggregation and Generalization, ACM Transactions on Database Systems, Vol. 2, No. 2 (1977), S. 105-133
  Download über Uni Wien Rechner/VPN: http://doi.acm.org/10.1145/320544.320546

J. M. Smith and D. C. P. Smith: Database Abstraction: Aggregation, Communications of the ACM, Vol. 20, Nr. 6, pp. 405-413, June 1977
Download über Uni Wien Rechner/VPN: http://doi.acm.org/10.1145/359605.359620

- Einige Weiterentwicklungen so z. B. Ende der 80er Jahre durch Wong und Katz.

- 1985 wurde IDEF1X als Standard zur ER-basierten Modellierung von der US Air Force veröffentlicht. 1994 wurde IDEF1X als U.S. Behördenstandard übernommen.

- Systemunabhängiges Datenmodell

- Dient in der konzeptionellen Phase der Anwendungsentwicklung der Verständigung zwischen Anwendern und Entwicklern, ermöglicht ein abstraktes Modell eines statischen Ausschnitts der realen Welt zu schaffen (WAS, aber nicht WIE)

- Dient in der Implementierungsphase als Grundlage für das Design von Datenbanken; weitgehend akzeptiertes Mittel zum konzeptionellen Datenbankentwurf

- Elemente des ER Modells finden sich in anderen Sprachen, auch in UML.

3.1.2 Nutzung des Entity-Relationship Modells

- ER-Modell wird vorzüglich bei der Erstellung von Datenbanken genutzt; dient oft zur Beschreibung von statischen Informationsmodellen.

- ER-Modell zur Konzeption der Datenbank dient als Grundlage für die Implementierung der Datenbank.

- Umsetzung der in der Realwelt erkannten Objekte und Beziehungen in ein Datenbank-Schema erfolgt dabei in mehreren Schritten:
  - Erkennen und Zusammenfassen von **Objekten** zu Entitätstypen durch Abstraktion (z. B. Die Kollegen Klas und Karagiannis und viele weitere zum Entitätstyp „Universitätsprofessor“).
  - Erkennen und Zusammenfassen von **Beziehungen** zwischen je zwei Objekten zu einem Beziehungstyp (z. B. Professor Klas koordiniert das Modul Grundl. d. Modellierung). Dieses führt zum Beziehungstyp „LV-Leiter koordiniert Modul“.
  - Bestimmung der **Kardinalitäten**, d. h. der Häufigkeit des Auftretens (z. B. wird ein Modul immer von genau einem LV-Leiter koordiniert und ein LV-Leiter darf mehrere Module koordinieren).
  - Bestimmung der relevanten **Attribute** der einzelnen Entitätstypen.

- Markierung bestimmter Attribute eines Entitätstyps als **identifizierende Attribute**, so genannte **Schlüsselattribute**.

- Weitere Schritte (nicht in dieser LV behandelt):

  - Um einen hohen **Gütegrad des Entwurfs** zu gewährleisten, sind Optimierungsschritte nötig: Durchführung des Prozesses der Normalisierung, um die Redundanz innerhalb der zu erstellenden Datenbank zu verringern und um die Datenintegrität zu erhöhen (siehe Datenbanksystemvorlesung).

  - **Generierung des Schemas** einer relationalen Datenbank mit all seinen Tabellen- und zugehörigen Felddefinitionen mit ihren jeweiligen Datentypen (siehe Datenbanksystemvorlesung).

3.1.3 Grundbegriffe

Grundlage der Entity-Relationship-Modelle ist die Typisierung von Objekten und deren Beziehungen untereinander:

- **Entität** (**Entity**): Objekt in der Realität, materiell oder abstrakt (zum Beispiel Student "Müller", Lehrveranstaltung "050039")

- **Entitätstyp**: Typisierung gleichartiger Entitäten (zum Beispiel Studierende, Lehrveranstaltung, Buch, Autor, Verlag)

- **Beziehung** (**Relationship**): semantische Beziehung zwischen zwei Objekten (zum Beispiel "Student Müller nimmt-teil-an Lehrveranstaltung "050039")

- **Beziehungstyp** (**Relationship-Type**): Typisierung gleichartiger Beziehungen (zum Beispiel Teilnahmebeziehung zwischen Studierenden und Lehrveranstaltung)

ACHTUNG: In der Regel wird bei einer Modellierung (in Diskussionen und Beispielen) mit konkreten Objekten gearbeitet (Entitäten und Beziehungen). Das Modell selbst hingegen besteht aber immer ausschließlich aus Entitätstypen und Beziehungtypen.

- **Attribut**: beschreibendes Merkmal einer Entität oder einer Beziehung (zum Beispiel Vorname und Nachname von Studierenden).

- **Kardinalität**: mögliche Anzahl der an einer Beziehung beteiligten Entitäten (zum Beispiel kann ein Student an mehreren Lehrveranstaltungen teilnehmen, während ein Modul von genau einem LV-Leiter koordiniert wird).

3.1.4 Sprachen zur Notation von ER-Modellen

- Die **grafische Darstellung** von Entitätstypen und Beziehungstypen wird **Entity-Relationship-Diagramm** (ERD) oder **ER-Diagramm** genannt.
  - Unterschiedliche Darstellungsformen.

- Vielzahl **unterschiedlicher Notationen**, die sich unter anderem in Klarheit, Umfang der grafischen Sprache, Unterstützung durch Standards und Werkzeuge unterscheiden. Kernaussage der ER-Diagramme aber jeweils nahezu identisch.

- Besondere - zum Teil historischer - Bedeutung:
  - **Chen-Notation** von Peter Chen
  - **IDEF1X** als langjähriger de-facto Standard bei U.S. amerikanischen Behörden.
  - **Bachman-Notation** von Charles Bachman als weit verbreitete Werkzeug-Diagramm-Sprache.
  - **Martin-Notation** (Krähenfuß-Notation) als weit verbreitete Werkzeug-Diagramm-Sprache (Information Engineering).
  - **Min-Max-Notation** als (kurzlebiger) ISO-Standard.
  - **UML** als aktueller Standard

3.1.5 Abstraktionskonzepte (allgemein)

- Abstraktion ist ein mentaler Prozess um einige Charakteristika und Eigenschaften einer Menge von Objekten zu deren Beschreibung auszuwählen und gleichzeitig andere, nicht relevante auszuschließen.

- Für die Datenmodellierung lassen sich 3 wesentliche Konzepte identifizieren:
  - Klassifikation
  - Aggregation
  - Verallgemeinerung bzw. Spezialisierung

- Eine **Klassifikation** dient der Definition bestimmter Konzepte als Klassen von Dingen bzw. Objekten mit gemeinsamen Eigenschaften.

  Eigenschaften werden in diesem Zusammenhang als Attribute bezeichnet

  Mathematisch geht es hierbei um Mengenbildung
  Ein Realwelt-Objekt kann auf viele Weisen klassifiziert werden. Welche Klassifikation gewünscht bzw. angemessen ist, muss im Einzelfall in Abhängigkeit von der Zielsetzung (Anwendung) entschieden werden.

- Eine **Aggregation** definiert eine neue Klasse aus anderen bereits existierenden, welche dann Komponenten repräsentieren, bzw. sie setzt bestehende Klassen zu neuen zusammen.

  Aus mathematischer Sicht handelt es sich um die Bildung kartesischer Produkte

- Eine **Generalisierung** bzw. **Spezialisierung** definiert eine Teilmengenbeziehung zwischen den Elementen verschiedener Klassen.

  Mathematisch geht es um Potenzmengenkonstruktion (bzw. Teilmengenbildung)

  Verallgemeinerungen bzw. Spezialisierungen haben eine Vererbungseigenschaft:
  Alle Konzepte (Attribute), welche für die allgemeinere Klasse definiert sind, werden an jede Teilmenge vererbt.

  Häufig (aber nicht immer) wird unter Spezialisierung eine disjunkte Zerlegung einer Klasse in Unterklassen verstanden.

3.1.6 Entities und Attribute

- Wohlunterscheidbare Dinge der realen Welt nennen wir im folgenden in Anlehnung an den englischen Sprachgebrauch ***Entities***. z.B. ein Auto.

- Einzelne Entities, welche “ähnlich”, “vergleichbar”, oder “zusammengehörig” sind, fassen wir zu einen ***Entity-Typ*** zusammen. z.B. alle Autos im System.

- Entities besitzen Eigenschaften, deren konkrete Ausprägungen wir als ***Werte*** bezeichnen. z.B. die Farbe meines Autos ist schwarz.

- Die Zusammenfassung aller möglichen bzw. zugelassenen Werte bezeichnet man als ***Wertebereich*** (**Domain**).

- Auf Ebene des Entity-Typs nennen wir die (bei allen Entities dieses Typs auftretenden) Eigenschaften ***Attribute***.

- Ein Attribut ordnet also jedem Entity des Entity-Typs einen Wert aus einem bestimmten Wertebereich (dem des Attributs) zu.

- Der Entity-Typ sowie dessen Attribute sind als zeitinvariant anzusehen. Diese zeitinvarianten Aspekte können grafisch dargestellt werden:
  - *Entity-Typen* werden als *Rechteck* dargestellt
  - *Attribute* werden als mit dem Rechteck verbundene *Kreise/Ellipsen* dargestellt

- Beispiel: Modell zu Bücher und Leser.

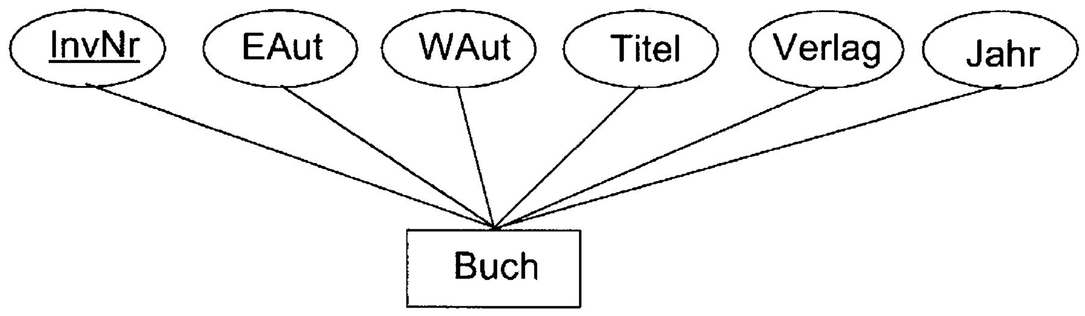


Abb. 3-1: Entity-Typ für Buchinformation

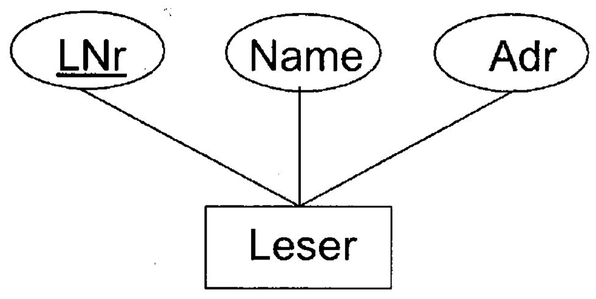


Abb. 3-2: Entity-Typ für Leserinformation

- Obige Beispiele sind unzureichend erfasst, da jedes Attribut als einwertig angenommen werden muss.

  De facto kann jedoch ein Buch mehrere Autoren haben und die Verlagsinformation ist aus einem Verlagsnamen und einem Verlagsort zusammengesetzt. Ebenso besteht ein Name typischerweise aus einem Vor- und einem Nachnamen und Adressen sind aus Straßen- und Ortsangaben zusammengesetzt.

- Weitere Typen von Attributen
  - ***Mehrwertige Attribute*** werden durch einen *Doppelkreis/Doppelellipse* dargestellt
  - Komponenten eines ***zusammengesetzten Attributs*** auch über Ellipsen, die verbunden werden.

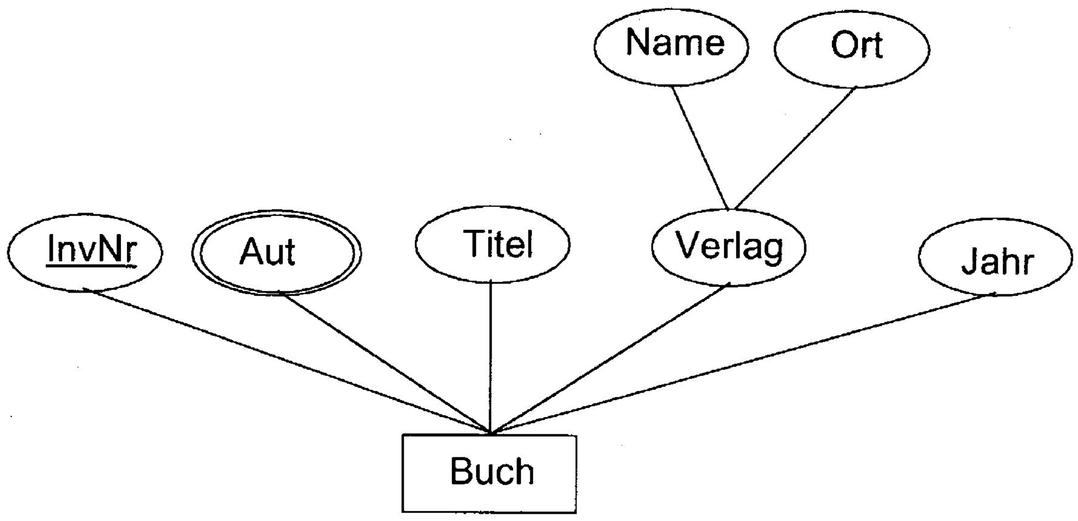


Abb. 3-3: Detaillierter Entity-Typ für Buchinformationen

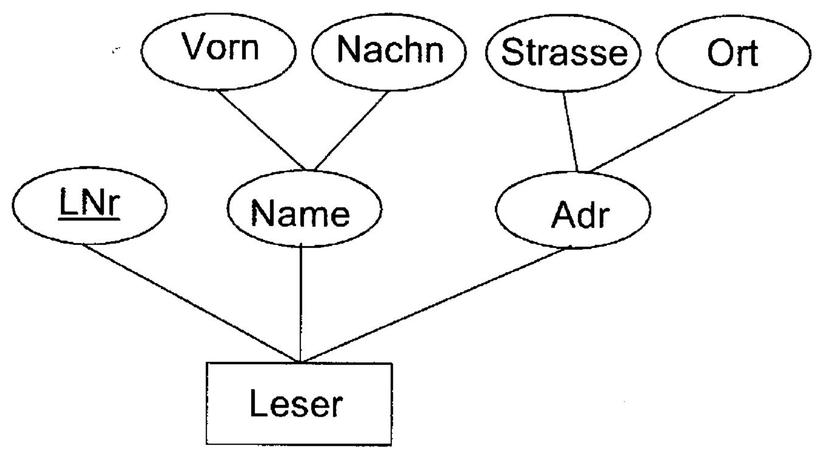


Abb. 3-4: Detaillierter Entity-Typ für Leserinformationen

- Es sei bemerkt, dass Mehrwertigkeit und Zusammensetzung nicht auf eine einmalige Anwendung beschränkt sein brauchen. Die Konzepte können theoretisch beliebig tief geschachtelt werden.

- Zur Beschreibung eines speziellen Entities ist häufig nicht die Angabe aller Attributwerte erforderlich. Es genügt oft für gewisse der einwertigen Attribute einen Wert anzugeben, durch welche das Entity eindeutig identifiziert wird.

- Attribute, die ein Entity eindeutig identifizieren, nennen wir **Schlüsselattribute** und ihre Zusammenfassung einen **Schlüssel** für das betreffende Entity

- Da wir die einzelnen Entities eines Entity-Typs als wohlunterscheidbar vorausgesetzt haben, kommt die Menge aller einwertigen Attribute des Typs immer als ein Schlüssel für den Typ in Frage. Im allgemeinen ist man jedoch in der Lage, bestimmte Attribute oder Attributkombinationen (aber nicht alle) als (zeitinvariante) Schlüssel auszuzeichnen.

- Es ist auch denkbar, dass sich für einen Entity-Typ mehr als ein Schlüssel bzw. genauer Schlüsselkandidat angeben lässt.

Beispiel:

Entity-Typ: Stadt
Attribute: PLZ, Name, Einwohner, Vorwahl

Schlüssel: PLZ oder Vorwahl

- Es wird stets ein Schlüssel als **Primärschlüssel** ausgezeichnet und in die Beschreibung eines Typs aufgenommen.

- Ein Schlüssel K ist immer eine **minimale, identifizierende Attributkombination**. Jede (echte) Obermenge von K ist ein Superschlüssel.

- **Definition**:
  Eine **Entity-Deklaration** hat die Form E = (X,K). Sie besteht aus einem Namen E, einem Format X und einem Primärschlüssel K, welcher aus (einwertigen) Elementen von X zusammengesetzt ist.

  Die Elemente eines Formats X werden dabei wie folgt notiert:
  (i) Einwertige Attribute: A
  (ii) Mehrwertige Attribute: {A}
  (iii) Zusammengesetzte Attribute: $A(B_1,...,B_k)$

  Nach gängigem Sprachgebrauch bezeichnet man eine Entity-Deklaration auch als **Entity-Typ** oder **Entity-Set**.

- **Beispiel**:
  Die Entity-Deklaration aus Abb. 3-3:
  Buch=((InvNr, {Autor}, Titel, Verlag(Name, Ort), Jahr), (InvNr))

  Die Entity-Deklaration aus Abb. 3-4:
  Leser=((LNr, Name(Vorn,Nachn), Adresse(Strasse,Ort)), (LNr))

3.1.7 Relationships

- Verschiedene Entity-Typen eines betrachteten Systems stehen im allgemeinen miteinander in Beziehung

  z.B.: Bücher werden von Lesern entliehen, sodass durch den Entlehnungsvorgang ein bestimmtes Buch mit einem bestimmten Leser in Beziehung gesetzt wird.

- An einer Beziehung sind Entities beteiligt

- Beziehungen können Attribute besitzen, welche spezielle Eigenschaften zum Ausdruck bringen, die erst durch das Herstellen der Beziehung relevant werden, jedoch für die einzelnen Entity-Typen bedeutungslos sind. Z.B.: Rückgabedatum der Beziehung "entliehen".

  Attribute können wie bei Entities einwertig, mehrwertig oder zusammengesetzt sein.

- Wie bei Entities unterscheidet man bei Relationships zwischen den zeitinvarianten Beschreibungen der Beziehungstypen und dem zeitveränderlichen Inhalt.

- **Definition**:
  Eine **Relationship-Deklaration** hat die Form
  R = (Ent,Y). Dabei ist R der Name der Deklaration ("Name der Beziehung"), und Ent die Folge der Namen der Entity-Deklarationen, zwischen denen eine Beziehung definiert werden soll, und Y ist eine (möglicherweise leere) Folge von Attributen (der Beziehung).

  Die Relationship-Deklaration wird auch als **Relationship-Typ** bzw. **Beziehungstyp** bezeichnet.

- **Beispiel**:
  Die Relationship-Deklaration aus Abb. 3-5:
  Ausleihe = ((Buch, Leser), (RDat))

Die Relationship-Deklaration aus Abb. 3-7:
liefert = ((Lieferant, Teil, Filiale), (Preis, Anz))

- Ein Beziehungstyp wird durch eine Raute, welche den Namen der Deklaration enthält, dargestellt. Diese wird durch Kanten mit den beteiligten Entity-Typen verbunden.

- Falls der Beziehungstyp eigene Attribute besitzt, werden diese durch Kreise/Ellipsen dargestellt und über Kanten mit der entsprechenden Raute verbunden.

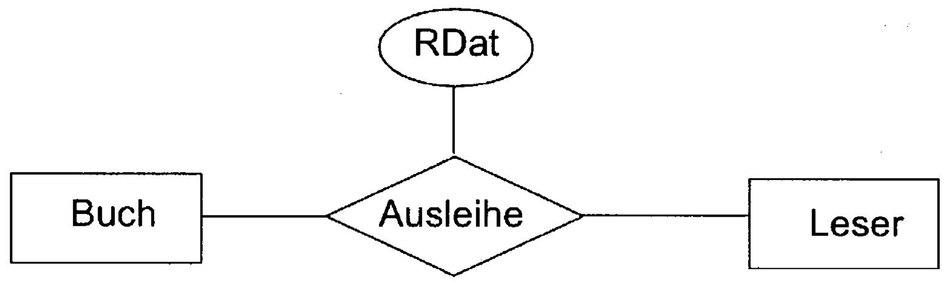


Abb. 3-5: Beziehung zwischen Büchern und Lesern

- Sei nun R = (Ent, Y) ein Beziehungstyp mit
  Ent = $(E_1,...,E_k)$. Dann heißt k die **Stelligkeit** oder der **Grad** von R.
  Es sei bemerkt, dass der Fall k = 2 ("binäre Beziehung") in praktischen Anwendungen der bei weitem häufigste Fall ist.

- Ein Entity kann jedoch auch mit sich selbst in Beziehung gesetzt werden. Man spricht auch von **rekursiven** oder **unären** Beziehungen. Auf jeder Seite der Beziehung übernimmt das Entity eine andere Rolle.

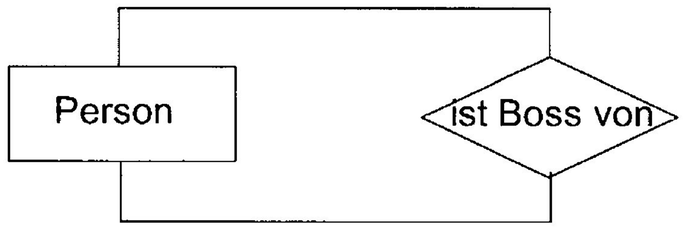


Abb. 3-6: Rekursive Beziehung zwischen Personen

- Jedoch können an einer Beziehung auch 3 (“ternäre Beziehung”) oder mehr (“n-äre Beziehungen”) Entity-Typen beteiligt sein.

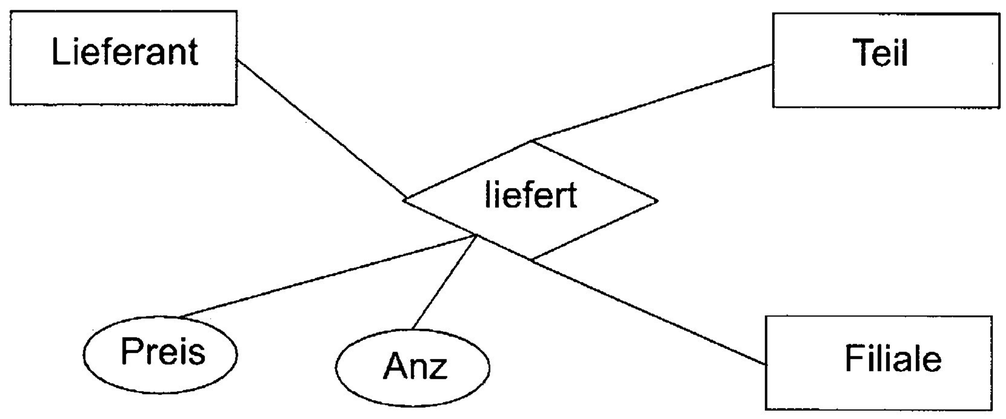


Abb. 3-7: ternäre Lieferantenbeziehung

- Beziehungstypen wird eine **Kardinalität** (Komplexität) zugeordnet, welche angibt, mit wievielen Entities des zweiten Typs (bei k = 2) ein bestimmtes Entity des ersten in einer konkreten Beziehung stehen kann, darf oder sogar muss.

- Ist R eine Relationship-Deklaration R = (Ent, Y) mit Ent = $(E_1, ..., E_k)$, so tritt ein spezielles Entity $e_i \in E_i^t$ (zum Zeitpunkt t) in einem Relationship-Set R^t mindestens m-mal, aber höchstens n-mal auf. Dies wird durch die Schreibweise “comp(R,E_i) = (m,n)” ausgedrückt. Für den Fall k = 2 gilt:

R ist vom Typ	**lies**	**falls gilt**		**Bedeutung**
		comp(R,E_1) $\in$	**comp(R,E_2) $\in$**	
1:1	[0 oder 1] zu [1 oder 0]	{(0,1), (1,1)}	{(0,1), (1,1)}	Jede Entität aus der ersten Entitätsmenge kann mit höchstens einer Entität aus der zweiten Entitätsmenge in Beziehung stehen, und umgekehrt.
1:n	[0 oder 1] zu beliebig vielen	{(0,∞), (1,∞)}	{(0,1), (1,1)}	Jede Entität aus der ersten Entitätsmenge kann mit beliebig vielen Entitäten aus der zweiten Entitätsmenge in Beziehung stehen. Jede Entität aus der zweiten Entitätsmenge kann mit höchstens einer Entität aus der ersten Entitätsmenge in Beziehung stehen.
n:m	beliebig viele zu beliebig vielen	{(0,∞), (1,∞)}	{(0,∞), (1,∞)}	Jede Entität aus der ersten Entitätsmenge kann mit beliebig vielen Entitäten aus der zweiten Entitätsmenge in Beziehung stehen, und umgekehrt.

- Die Kardinalität wird an den Kanten der Beziehung angezeigt.
- Eine Beziehung ist vom Typ 1:1, falls jedes Entity vom Typ E_1 mit höchstens einem Entity vom Typ E_2 in Beziehung steht und umgekehrt.

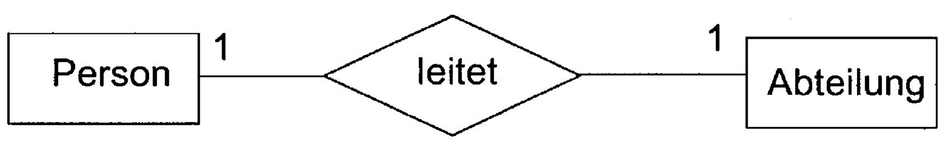


Abb. 3-8: 1:1-Beziehung zwischen Abteilung und ihrem Leiter

- Eine Beziehung ist vom Typ 1:n, falls ein Entity vom Typ E_1 mit $n \geq 0$ Entities vom Typ E_2 in Beziehung steht, andererseits jedoch jedes Entity vom Typ E_2 mit höchstens einem vom Typ E_1 assoziiert ist.

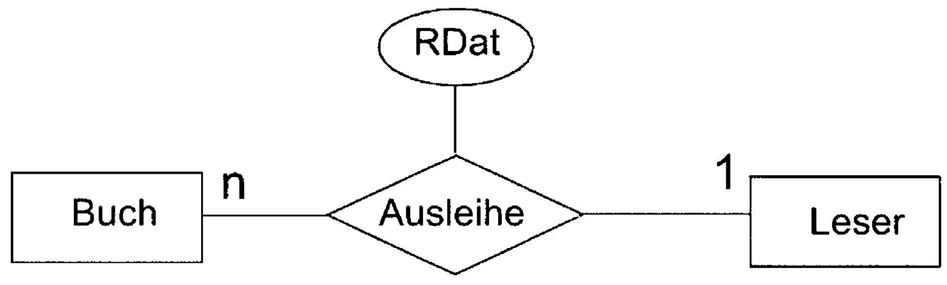


Abb. 3-9: Beziehung zwischen Büchern und Lesern

- Eine Beziehung vom Typ n:m stellt keine Restriktionen an die Entity-Paare eines Beziehungstyps.

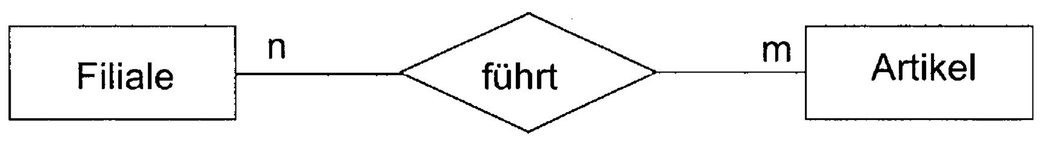


Abb. 3-10: Beziehung zwischen Filialen und Artikeln

- Es können zwischen den selben Entity-Typen auch mehrere unterschiedliche Beziehungstypen bestehen.

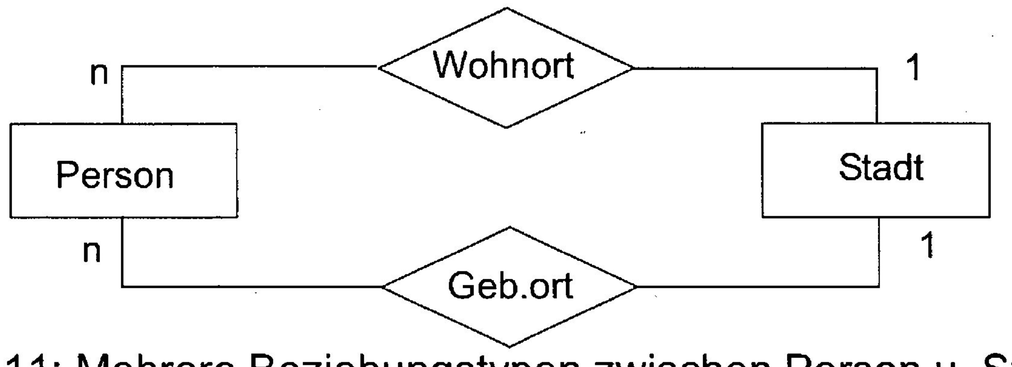


Abb. 3-11: Mehrere Beziehungstypen zwischen Person u. Stadt

- Für den Fall $k \geq 3$, gibt die Kardinalität an, mit wievielen Entities eines bestimmten Typs eine bestimmte Kombination von Entities der restlichen Typen (eine Entity pro Typ) in einer konkreten Beziehung stehen kann, darf oder sogar muss.

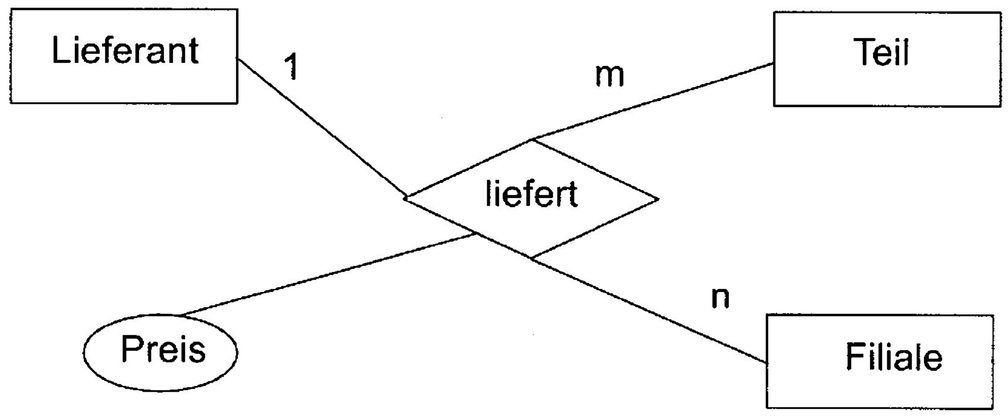


Abb. 3-12: ternäre Lieferantenbeziehung

3.1.8 IS-A-Beziehungen

- Es gibt den Fall, dass die Entities eines Typs nicht (allein) über ihre Attribute unterschieden werden, sondern über ihre Beziehung zu Entities eines anderen Typs.

- Beispiel:

  Angestellter = ((AngNr, Name, Adresse, Beruf, Gehalt),(AngNr))

  Für Piloten wird die Erfahrung in Flugstunden gemessen und sie besitzen eine Fluglizenz. Techniker gehören hingegen gewissen Technikerteams an. In beiden Fällen gibt es Attribute, welche für bestimmte Angestellte der Gesellschaft, aber nicht für alle relevant sind. In diesem Fall ist es sinnvoll, jeweils separate Entity-Deklarationen vorzunehmen:

  Pilot = ((AngNr, Std, Liz) , (AngNr))
  Techniker = ((AngNr, TNr) , (AngNr))

  In beiden Fällen handelt es sich um Spezialisierungen von Angestellter, welche zu diesen Deklarationen in einer sogenannten IS-A-Beziehung steht. Ein konkreter Pilot kann also als Entity vom Typ Angestellter und als Entity vom Typ Pilot aufgefasst werden.

- Eine konkrete Entity eines Subtyps (Spezialisierung) kann auch als Entity des Supertyps (Generalisierung) aufgefasst werden.

- Alle Attribute des Supertyps werden an die der Subtypen vererbt. Es ist nicht notwendig in einer grafischen Darstellung sämtliche Attribute eines Subtyps anzuzeigen, sondern es reichen die neuen Attribute.

- **Definition**:
  Sind $E_1=(X_1,K_1)$ und $E2=(X_2,K_2)$ zwei Entity-Deklarationen, so besteht zwischen diesen eine **IS-A-Beziehung** der Form E_1 IS-A E_2, falls gilt:
  (i) Alle Elemente von X_2 kommen in X_1 vor;
  (ii) zu jedem Zeitpunkt t gilt: Für jedes $e_1 \in E_1^t$ existiert ein $e_2 \in E_2^t$ mit $e_1(A) = e_2(A)$ für jedes Attribut $A \in X_2$.

  Man schreibt auch kurz $E_1 \subseteq E_2$.

- Die (explizite) Bedingung (i) zusammen mit der (impliziten) Forderung, dass K_1 ein Schlüssel für X_1 (also für die Folge aller Attribute) ist, folgende Implikationen besitzen kann:
  - Es gilt $K_1 = K_2$; häufigster Fall
  - K_2 ist eine Teilfolge vom K_1; es gibt für die Spezialisierung neue Attribute, welche zur Identifikation eines Entities benötigt werden.

- Zur grafischen Darstellung von IS-A-Beziehungen verwenden wir wieder Rauten, in die jetzt "IS-A" eingetragen wird. Diese werden mit ungerichteten Kanten mit der oder den Subtypen und mit einer gerichteten Kante zum Supertyp verbunden.

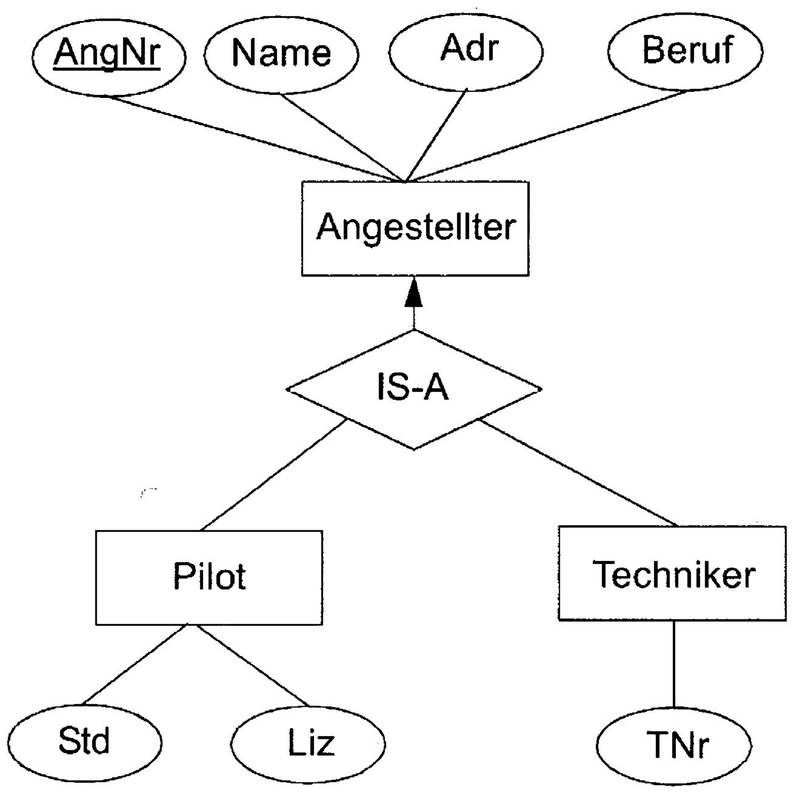


Abb. 3-13: IS-A-Beziehung zw. Angestellten, Piloten und Technikern

- Die Auszeichnung einer Reihe von Spezialisierungen einer einzelnen Entity-Deklaration kann
  - total (t) oder partiell (p) sein
  - disjunkt (d) oder nicht-disjunkt (n) sein

  Es ergeben sich 4 verschiedene Kombinationen

- **Definition**:
  Es seien E, E_1, ..., E_k Entity Deklarationen, $k \geq 2$, und es gelte, dass alle E_i, $1 \leq i \leq k$ zu derselben Spezialisierung von E gehören. Die IS-A-Beziehung heißt
  - **total**, falls gilt: $(\forall t) \cup_{i=1}^{k} E_i^t = E^t$

    andernsfalls **partiell**
  - **disjunkt**, falls gilt: $(\forall t)\ (\forall i,j \in \{1,...k\}, i \neq j) E_i^t \cap E^t = 0$

    andernfalls **nicht-disjunkt**

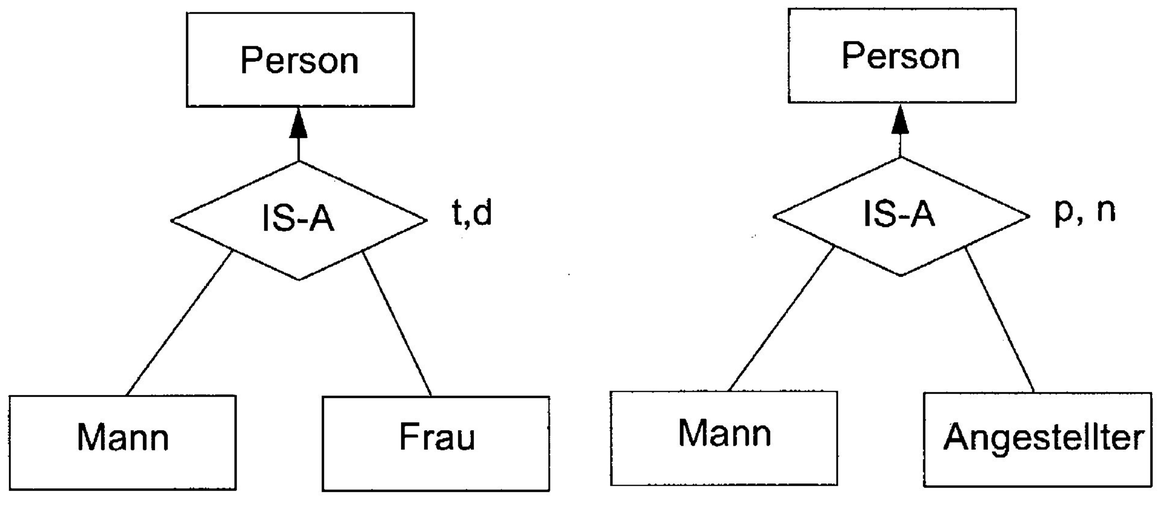
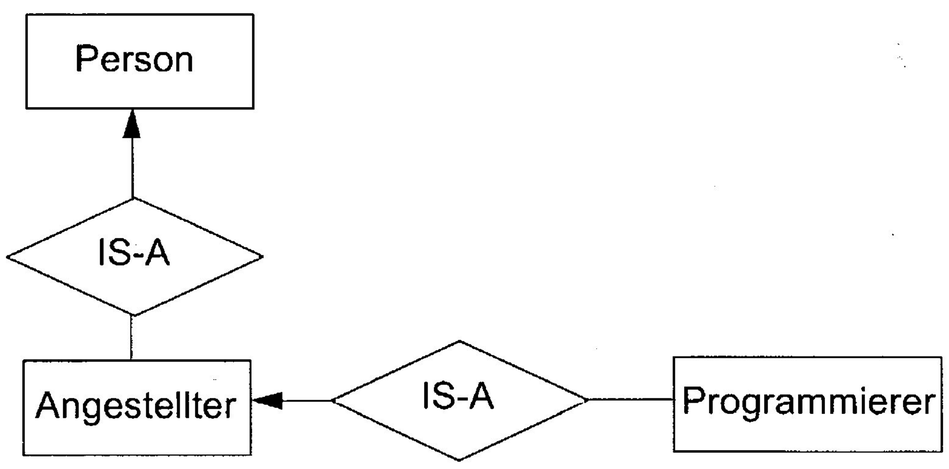


Abb. 3-14: Arten von IS-A-Beziehungen

- Spezialisierungen bzw. Generalisierungen können auch über mehrere Ebenen vorgenommen werden



3.1.9 Beispiel

Das Beispiel behandelt einen Ausschnitt einer konzeptionellen Modellierung für eine Fluggesellschaft. Die Gesellschaft will Daten über Personen, insbesondere Passagiere und die Angestellten, ihre Flugzeuge und die durchgeführten Flüge speichern. Wir betrachten zunächst die Personen genauer: Jede Person habe zur Identifikation eine Nummer, einen Namen, eine Adresse und ein Geburtsdatum. Ein Passagier hat zusätzlich ein Geschlecht und eine Angabe, ob er Raucher ist oder nicht. Angestellte haben eine Angestelltennummer, einen Beruf und ein Gehalt. Unter ihnen sind die Piloten und die Techniker ausgezeichnet.
Bei der Modellierung von Flugzeugen unterscheiden wir zwischen Flugzeugen, welche die Gesellschaft besitzt, gekennzeichnet durch Seriennummer, Kaufdatum, die Summe der bisherigen Flugstunden, den Typ des Ausbaus, eine Menge in jüngster Zeit vorgekommener Störfälle und die zuletzt durchgeführte Wartung und solchen Angaben, die für alle Maschinen eines Typs gültig sind, wie Hersteller, Typ, Länge, Reisegeschwindigkeit, Sitzanzahl, Wartungsintervalle und Reichweite. (So hat z.B. jeder Airbus 330-400 die gleiche Reisegeschwindigkeit; falls die Gesellschaft jedoch mehrere Maschinen dieses Typs besitzt, so sind diese über ihre Seriennummer unterscheidbar.)
Analog unterscheiden wir zwischen bestimmten, im allgemeinen regelmäßig verkehrenden Flügen (z.B. OS 873 von Wien nach Athen) und den einzelnen, tatsächlich an einem bestimmten Datum durchgeführten Abflügen (OS 873 am 28. 3. 2001), welche unter der Leitung eines bestimmten Piloten mit einer bestimmten Maschine der Gesellschaft durchgeführt wird und Passagiere an Bord hat. Die Unterscheidung zwischen "realen" und "virtuellen" Maschinen bzw. Flügen ermöglicht auch eine Unterscheidung zwischen der konkret ausgeführten und der möglichen Tätigkeit eines Piloten: Ein Pilot kann z.B. nicht nur eine spezielle, sondern jeden Airbus 330-400 fliegen; real eingesetzt wird er hingegen nur bei speziellen Flügen, aber nicht notwendig bei jedem Flug mit der gleichen Bezeichnung. Analog sind die Beziehungen "kann warten" und wartet zwischen Technikern und Fluggerät bzw. Flugzeugen zu verstehen.

Es sei bemerkt, dass die Lösung keinen Anspruch auf Vollständigkeit erhebt.

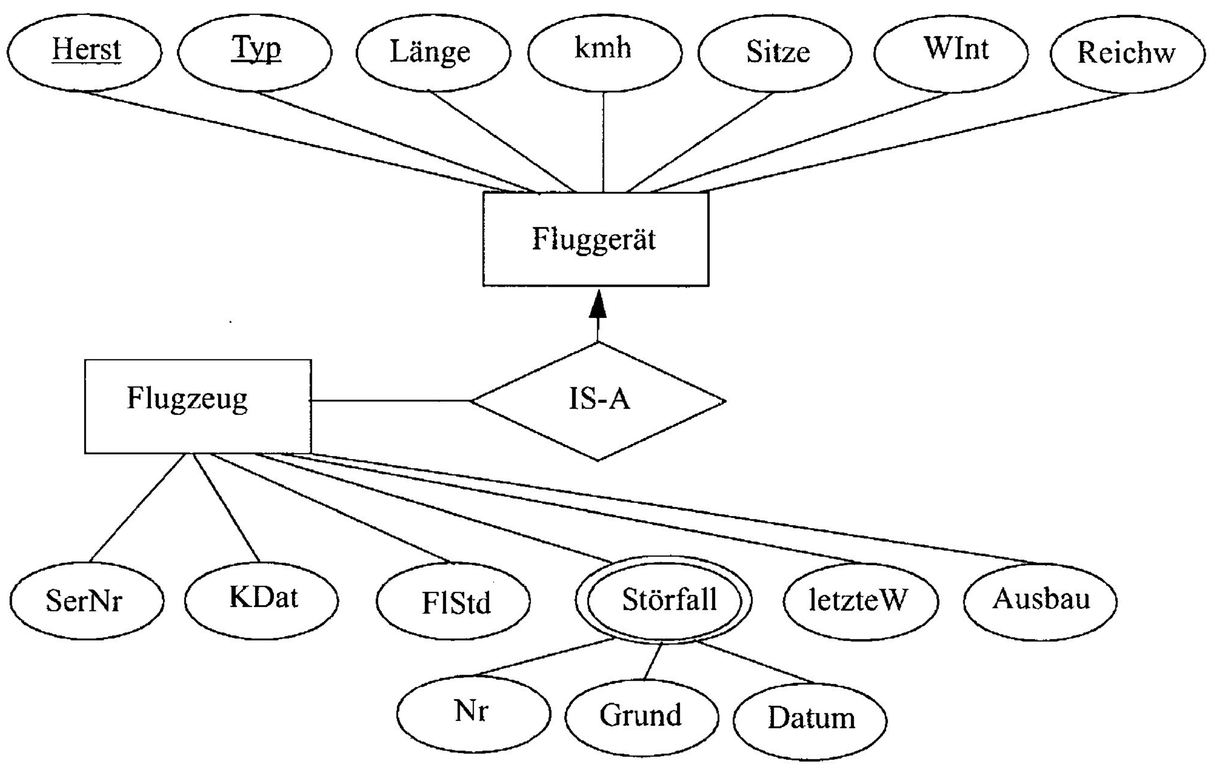


Abb. 3-15: Erstes Teildiagramm für die Fluggesellschaft

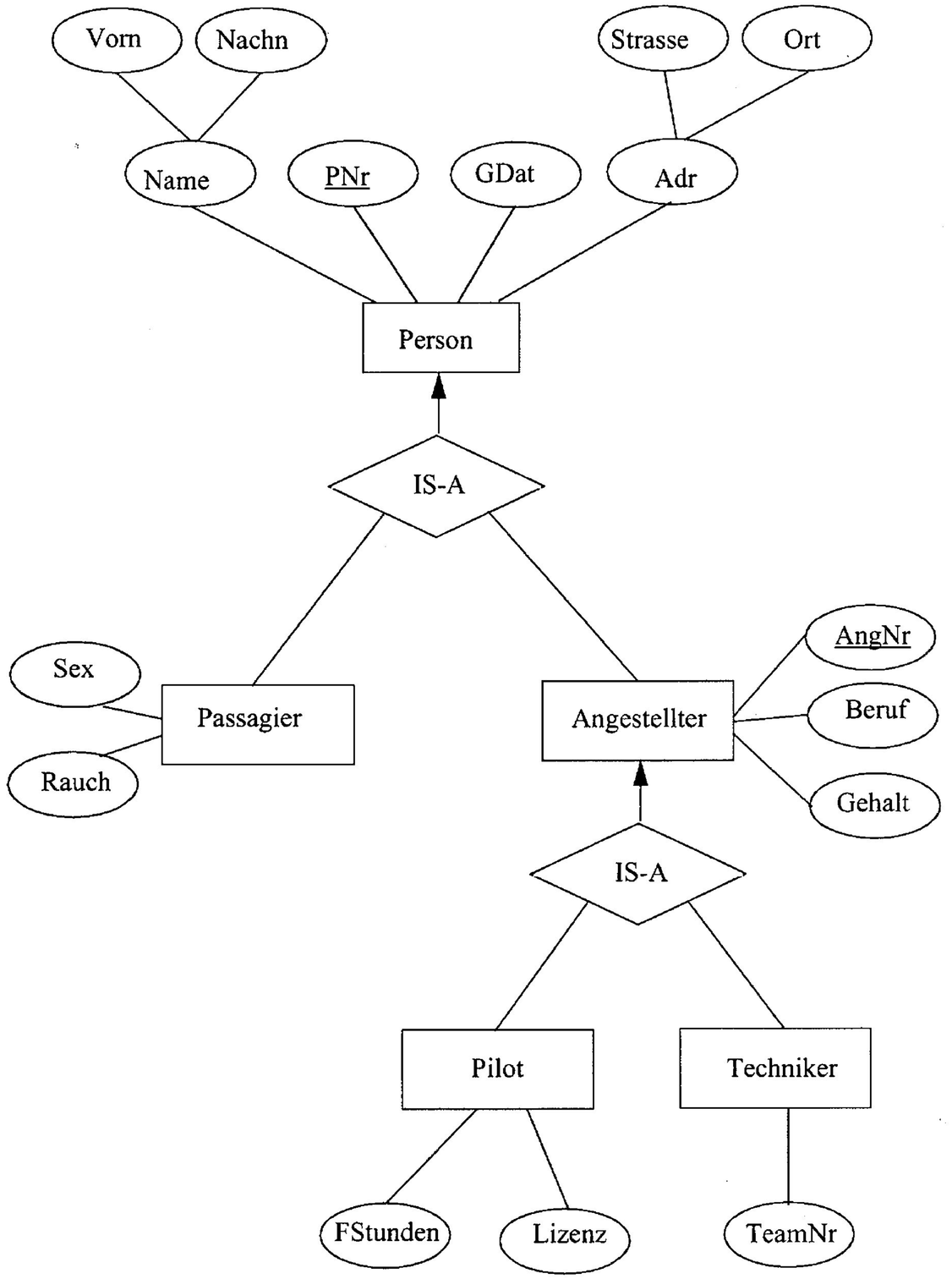


Abb. 3-16: Zweites Teildiagramm für die Fluggesellschaft

© Prof. Dr. Wolfgang Klas, Prof. Dr. Dimitris Karagiannis, SS 2011 - Modellierung
3 Das Entity-Relationship-Modell

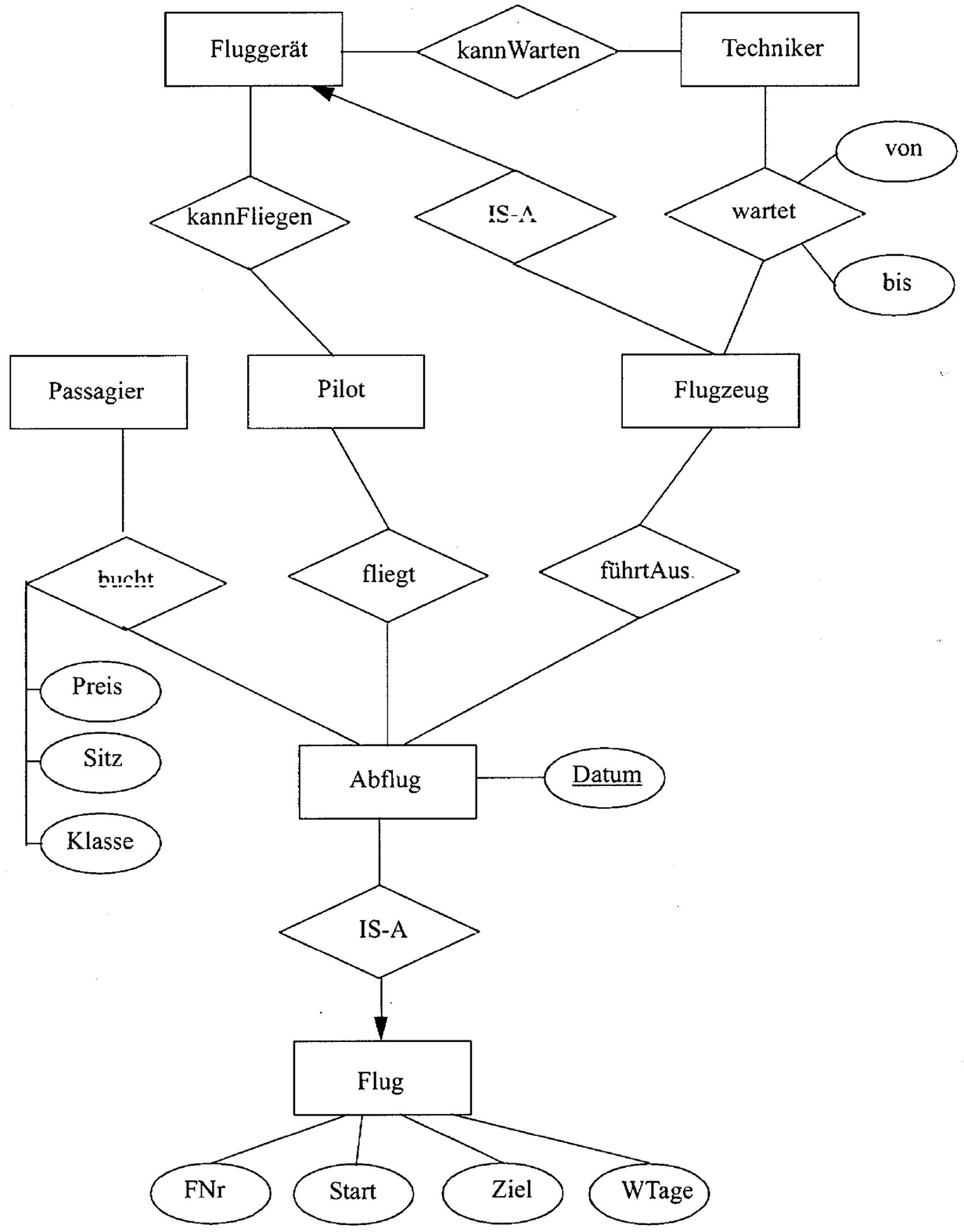


Abb. 3-17: Drittes Teildiagramm für die Fluggesellschaft

© Prof. Dr. Wolfgang Klas, Prof. Dr. Dimitris Karagiannis, SS 2011 - Modellierung
3 Das Entity-Relationship-Modell

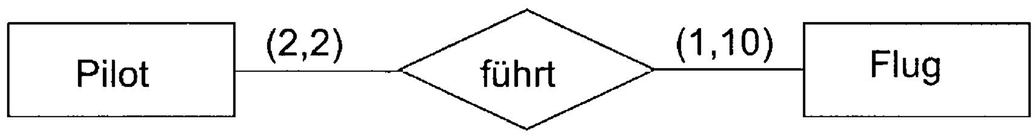
3.2 Erweiterungen und Spezialfälle der ER-Modellierung

3.2.1 Min,Max-Notation

- Bei der Min,Max-Notation für einen Beziehungstyp wird nicht nur die maximale Komplexität in der Kardinalität der beteiligten Entity-Typen angegeben, sondern die Kardinalität wird in Form eines Intervalls [u,o] angegeben.

- Eine Beziehung hat die Kardinalität (u, o), wenn zu jeder Entität aus der Entitätsmenge X mindestens u, aber höchstens o verschiedene Entitäten aus der Entitätsmenge Y in Beziehung gebracht werden können

- Die Grenzen des Intervalls bedeuten daher, dass zwischen u und o Entitäten des betroffenen Entity-Typs mit dem in Relation stehenden Entity-Typ (bei binären Relationen) in Beziehung stehen können.



- Notation der Kardinalität (u, o) als Intervall [u, o]
  - 0 steht für kein Vorkommen
  - 1 steht für einmaliges Vorkommen
  - * steht für beliebig oftes Vorkommen (0.. ∞)

- Vorteil: sowohl Unter- als auch Obergrenzen können präzise ausgedrückt werden

- Nachteil: keine Modellierung von Beziehungen, an denen drei Entity-Typen beteiligt sind. Daher wird die min,max-Notation oft eingeschränkt im Sinne der Chen-Notation verwendet

- Für die Überführung in Relationenschemata gilt, dass eine Oberschranke $o \geq 2$, wie ein "n" zu behandeln ist (siehe dazu Datenbanksystem-Vorlesung).

- Achtung: Manche Autoren verwenden bei der Min/Max-Notation eine Umkehrung der Kantenbeschriftung.

3.2.2 Modifizierte Chen-Notation (MC-Notation)

- MC-Notation ist eine Erweiterung der Chen-Notation:
  - Aussage "kein oder ein Element" mit dem Buchstaben c (choice, can)
  - Aussage "ein oder mehr Element(e)" mit dem Buchstaben m (must, multiple)

  angegeben wird.
  (Daher wird MC manchmal auch als Must-Can interpretiert.)

- **1:1** (1 zu 1): Jede Entität der ersten Entitätsmenge steht mit genau einer Entität der zweiten Entitätsmenge in Beziehung, und umgekehrt.

- **1:c** (1 zu (0 oder 1)): Jede Entität der ersten Entitätsmenge kann mit höchstens einer Entität der zweiten Entitätsmenge in Beziehung stehen. Jede Entitiät der zweiten Enititätsmenge steht mit genau einer Entität der ersten Entitätsmenge in Beziehung.

- **1:m** (1 zu (mindestens 1)): Jede Entität der ersten Entitätsmenge steht mit mindestens einer Entität der zweiten Entitätsmenge in Beziehung. Jede Entitiät der zweiten Enititätsmenge steht mit genau einer Entität der ersten Entitätsmenge in Beziehung.

- **1:mc** (1 zu (beliebig viele)): Jede Entität der ersten Entitätsmenge kann mit beliebig vielen Entitäten der zweiten Entitätsmenge in Beziehung stehen. Jede Entitiät der zweiten Enititätsmenge steht mit genau einer Entität der ersten Entitätsmenge in Beziehung.

- **c:c** ((1 oder 0) zu (0 oder 1); entspricht 1:1 in Chen-Notation): Jede Entität der ersten Entitätsmenge kann mit höchstens einer Entität der zweiten Entitätsmenge in Beziehung stehen, und umgekehrt.

- **c:m** ((0 oder 1) zu (mindestens 1)): Jede Entität der ersten Entitätsmenge steht mit mindestens einer Entität der zweiten Entitätsmenge in Beziehung. Jede Entitiät der zweiten Enititätsmenge kann mit höchstens einer Entität der ersten Entitätsmenge in Beziehung stehen.

- **c:mc** ((0 oder 1) zu (beliebig viele); entspricht 1:n in Chen-Notation): Jede Entität der ersten Entitätsmenge kann mit beliebig vielen Entitäten der zweiten Entitätsmenge in Beziehung stehen. Jede Entitiät der zweiten Enititätsmenge kann mit höchstens einer Entität der ersten Entitätsmenge in Beziehung stehen.

- **m:m** ((mindestens 1) zu (mindestens 1)): Jede Entität der ersten Entitätsmenge steht mit mindestens einer Entität der zweiten Entitätsmenge in Beziehung, und umgekehrt.

- **m:mc** ((mindestens 1) zu (beliebig viele)): Jede Entität der ersten Entitätsmenge kann mit beliebig vielen Entitäten der zweiten Entitätsmenge in Beziehung stehen. Jede Entitiät der zweiten Enititätsmenge steht mit mindestens einer Entität der ersten Entitätsmenge in Beziehung.

- **mc:mc** ((beliebig viele) zu (beliebig vielen)); entspricht m:n in Chen-Notation): Jede Entität der ersten Entitätsmenge kann mit beliebig vielen Entitäten der zweiten Entitätsmenge in Beziehung stehen, und umgekehrt.

3.2.3 Gegenüberstellung Min-Max-Notation, Chen-Notation und MC-Notation

Chen-Notation: Entität 1 ← 1:n → Entität 2
MC-Notation: Entität 1 ← c:mc → Entität 2
Min-Max-Notation: Entität 1 ← (0,1) ——— (0,*)→ Entität 2

Entität 1	Chen-Notation	MC-Notation	Entität 2
(0,1)	1:1	c:c	(0,1)
(0,1)	1:n	c:mc	(0,*)
(0,1)	---	c:m	(1,*)
(0,*)	m:n	mc:mc	(0,*)
(1,1)	---	1:c	(0,1)
(1,1)	---	1:mc	(0,*)
(1,1)	---	1:1	(1,1)
(1,1)	---	1:m	(1,*)
(1,*)	---	m:mc	(0,*)
(1,*)	---	m:m	(1,*)

Tabelle 3-1: Vergleich der Notationen zur Kardinalität

3.2.4 Weak Entity-Typ (Schwache Entity)

- Es gibt Situationen, in denen einem Entity-Typ **nicht genügend viele Attribute zur Bildung eines Schlüssels** zugewiesen werden können. Diese Entity-Typen bezeichnet man als Weak Entity-Typen bzw. schwache Entity-Typen.

- Ein Weak Entity kann nicht isoliert existieren, sondern ist stets über eine Existenzbedingung mit einem gewöhnlichen Entity verbunden.

- Die Schlüsselattribute eines Weak Entity-Typs werden durch den Schlüssel des übergeordneten Typs vervollständigt.

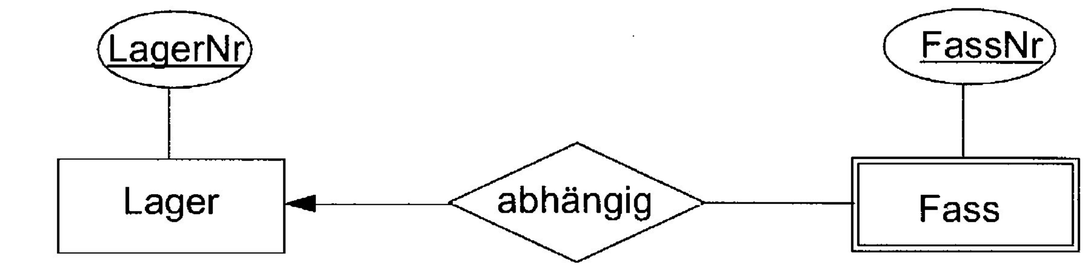
- Damit erbt der Weak Entity-Typ die Schlüsselattribute des übergeordneten Typs und erweitert diese um spezifische Schlüsselattribute.

- Die Beziehung eines Weak Entity-Typ ist verwandt zur IS-A-Beziehung, hat aber eine andere Semantik.

- Darstellung des Weak Entity-Typs durch doppelt umrandetes Rechteck.

- Beispiel:

Annahme: Die Fassnummern sind generell nicht eindeutig
jedoch sind die Fässer innerhalb eines Lagers eindeutig numeriert.



Lager(LagerNr, ...)

Fass(LagerNr, FassNr, ...)

3.2.5 Uminterpretierter Beziehungstyp

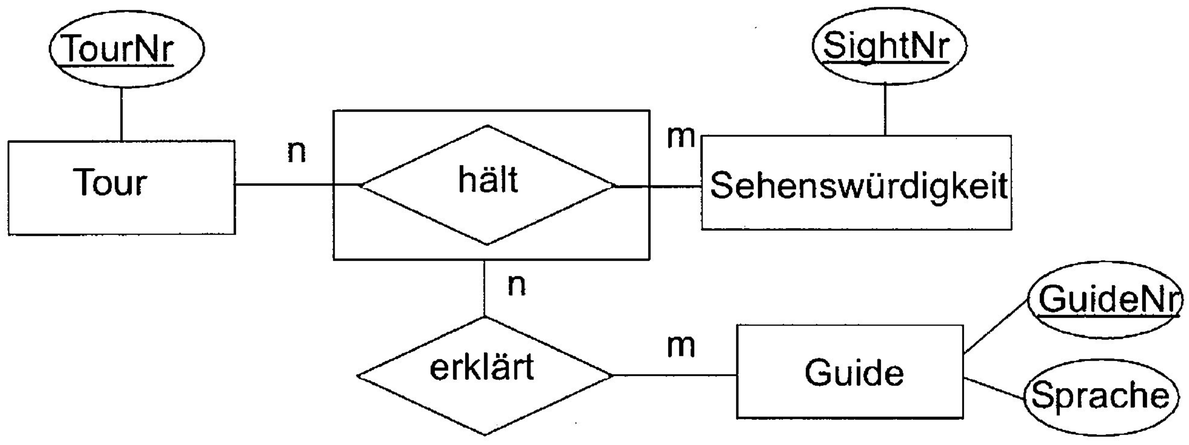
- Im herkömmlichen ER-Modell sind nur Beziehungs-Typen zwischen Entity-Typen möglich.

- In bestimmten Anwendungen kann es erforderlich werden, Beziehungen auch zwischen Beziehungstypen selbst zu modellieren.

- Dabei wird ein Beziehungs-Typ, der mit anderen Entity-Typen (bzw. Beziehungs-Typen) in Beziehung gesetzt werden soll, uminterpretiert. D.h. der Beziehungs-Typ wird in der Folge wie ein Entity-Typ behandelt.

- Zur grafischen Notation wird ein Rechteck über die Raute des Beziehungstyps geschlossen.
  Kanten, die zur Raute gehen, markieren eine Beteiligung am Beziehungstyp.
  Kanten, die zum Rechteck gehen, markieren eine Beteiligung des Uminterpretierten Beziehungstyps an einer anderen Beziehung

- Beispiel:

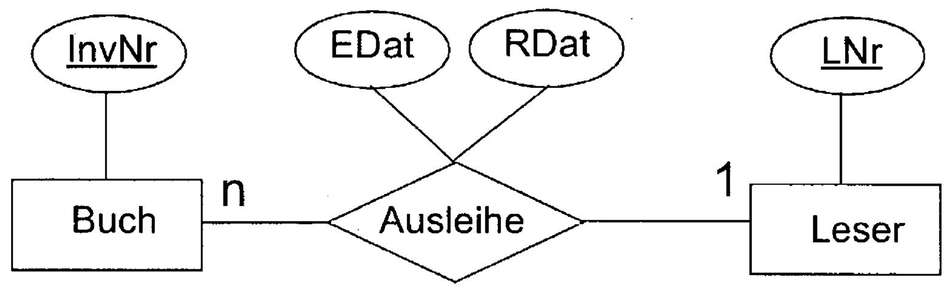


3.2.6 Historische Daten

- Für sehr viele Anwendungen ist es entscheidend, ob historische Daten gespeichert werden müssen oder nicht.

- Beispielsweise ist es entscheidend, ob bei einer Bibliothek die Daten der Leihe über zurückgegebene Bücher erhalten bleiben müssen.

- Betrachten wir die Ausleihrelation, die bislang in diesen Unterlagen verwendet wurde und ergänzen wir sie um das Entlehndatum:



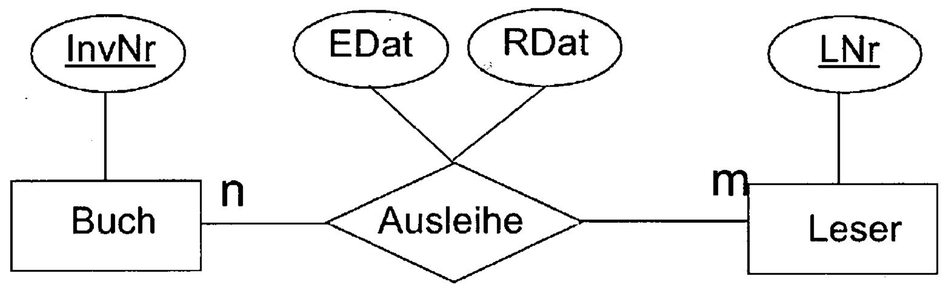
- Beispielsammlung für Ausleihe für den Fall, dass nur aktuelle Ausleihen gespeichert werden:

InvNr	LNr	EDat	RDat
1024	125	12.03.2007	12.05.2007
3700	125	12.03.2007	12.05.2007
5045	170	28.02.2007	28.04.2007
8345	131	09.03.2007	09.05.2007

- Beispielsammlung für Ausleihe für den Fall, dass auch historische Ausleihen gespeichert werden sollen:

InvNr	LNr	EDat	RDat
1024	125	12.03.2007	12.05.2007
3700	125	12.03.2007	12.05.2007
7830	125	01.08.2005	01.10.2005
5540	182	04.01.2007	04.03.2007

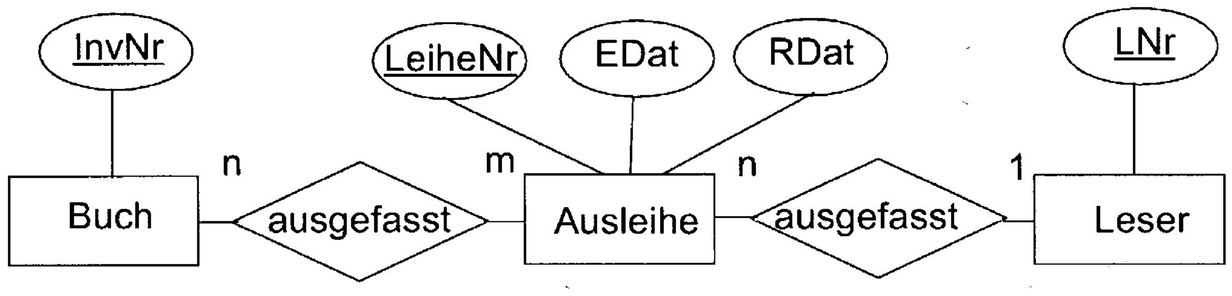
- Für die Speicherung historischer Ausleihen ist eine n:m-Beziehung nötig



- Beispielsammlung für Ausleihe für den Fall, dass ein Leser ein Buch öfters ausborgen kann:

InvNr	LNr	EDat	RDat
1024	125	12.03.2007	12.05.2007
3700	125	12.03.2007	12.05.2007
3700	125	12.01.2007	12.03.2007
7830	125	01.08.2005	01.10.2005

- Um ein wiederholtes Ausleihen eines Buchs zu speichern, wird die Ausleihe zur Entität, die mit Leser und Buch in Beziehung steht:



Literatur und Quellen:

- Peter Pin-Shan Chen: *The Entity-Relationship Model - Toward a Unified View of Data.* In: ACM Transactions on Database Systems, 1/1/1976, ACM-Press, ISSN 0362-5915, S. 9-36
  - Download über Uni Wien Rechner/VPN: http://doi.acm.org/10.1145/320434.320440

- J.M. Smith, D.C.P. Smith: *Database Abstractions: Aggregation and Generalization*, ACM Transactions on Database Systems, Vol. 2, No. 2 (1977), S. 105-133
  - Download über Uni Wien Rechner/VPN: http://doi.acm.org/10.1145/320544.320546

- J. M. Smith and D. C. P. Smith: *Database Abstraction: Aggregation*, Communications of the ACM, Vol. 20, Nr. 6, pp. 405-413, June 1977
  - Download über Uni Wien Rechner/VPN: http://doi.acm.org/10.1145/359605.359620

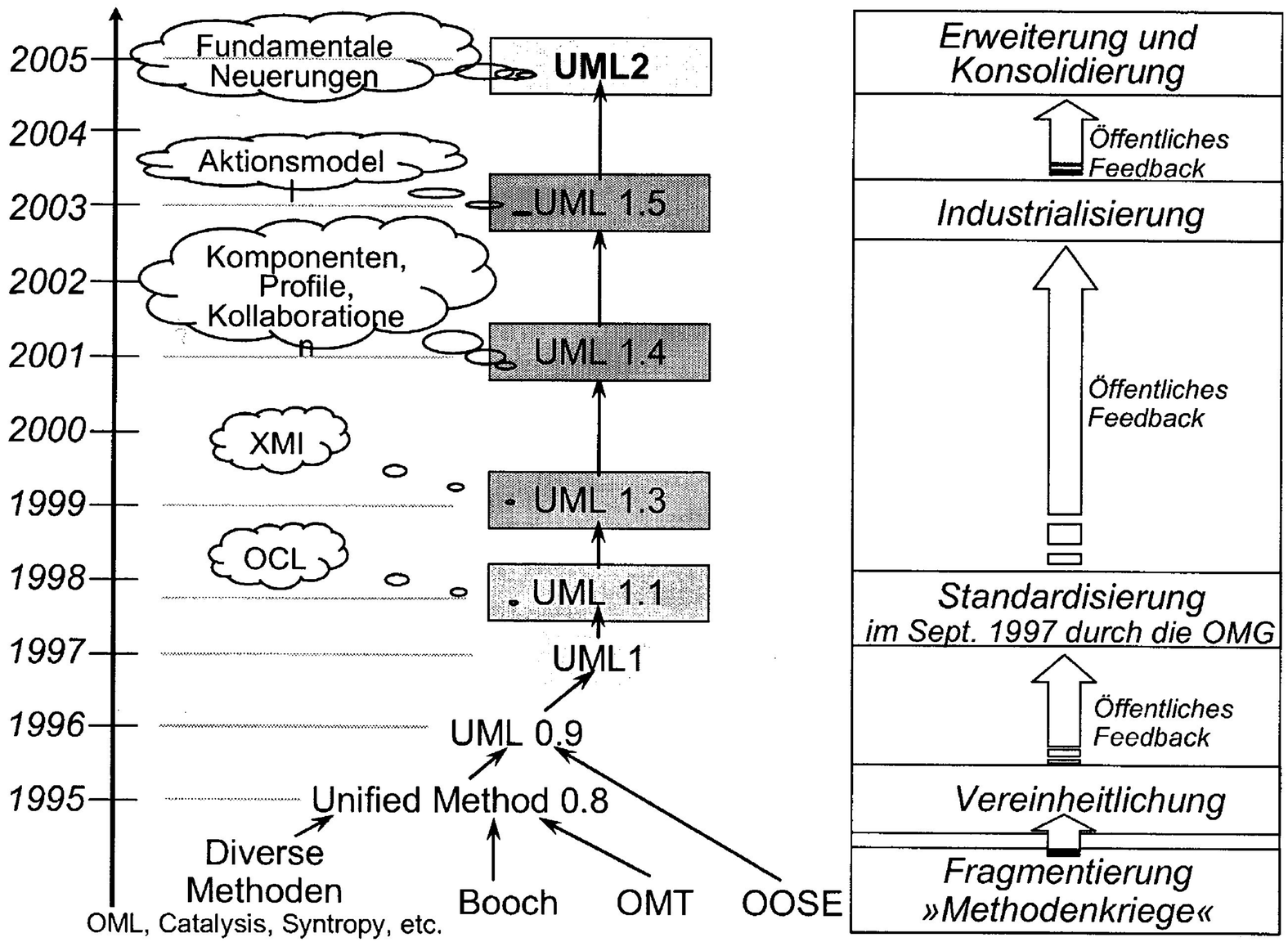
4 Objektorientierte Modellierung

Inhalt

4.1 Entwicklung objektorientierter Modellierung ..... -1
4.2 Grundlegende Konzepte der objektorientierten Modellierung ..... -3
4.3 Umsetzung objektorientierter Modellierung in UML ..... -4
4.3.1 Wichtige UML Schemata für statische Aspekte ..... -6
4.3.2 Wichtige UML Schemata für Dynamische Aspekte ..... -7
4.3.3 Wichtige UML Schemata für softwareorganisatorische Aspekte ..... -8
4.3.4 Zusammenspiel wichtiger Schemata in UML ..... -9
4.3.5 Modelle, Diagramme und Werkzeuge ..... -10
4.3.6 UML 2.0 Diagrammtypen ..... -11

4.1 Entwicklung objektorientierter Modellierung

- Objektorientierte Konzepte wurden erstmals in den 60iger Jahren in der Programmiersprache SIMULA eingesetzt.
- Verbreitung objektorientierter Programmiersprachen erst durch Smalltalk, Ada, C++ und Java (siehe Graphik in Abschnitt “2.3.4 Objektorientierte Modelle”).
- Objektorientierte Analyse und Entwurfsmethoden ab Mitte 80iger.
- Einem Methodenkrieg folgt 1996 Aufruf der Object Management (OMG) zur Schaffung eines Modellierungsstandards (siehe Graphik in Abschnitt “2.3.4 Objektorientierte Modelle”).
- 1998 wird die Unified Modeling Language (UML) 1.1 als Modellierungsstandard von der OMG akzeptiert.
- Seither Erweiterung von UML 1.1 und Etablierung von UML 2.0 als zentrale, weit akzeptierte Modellierungssprache (siehe nachfolgende Graphik).



4.2 Grundlegende Konzepte der objektorientierten Modellierung

Die Welt wird durch die “Objekt-Brille” gesehen

- Objekte sind eindeutig identifizierbare Einheiten, z.B. “der Student Karl”, “der Ablauf der LVA Modellierung”
- Objekte stehen zueinander in Beziehung, z.B. Karl “nimmt teil an” LVA Modellierung; Objekte können in Teilobjekte untergliedert sein, z.B. “die 1. Vorlesungseinheit von LVA Modellierung”
- Objekte befinden sich im Laufe ihres Lebenszyklus in unterschiedlichen Zuständen, z.B. Karl “übt” und hält derzeit beim “Punktestand 25”, und weisen Verhalten auf, z.B. Karl “löst” die 1. Übungsaufgabe, dann die 2., …
- Objekte können interagieren, z.B., Karl stellt die Frage “Wann ist Prüfung?” an den Professor; an einem Verhalten können damit mehrere Objekte teilnehmen

4.3 Umsetzung objektorientierter Modellierung in UML

- Mit UML definiert man **Arten** von Objekten, Interaktionsszenarien, Verhaltensweisen, etc. (**Schematisierung**)

- Einzelne Objekte und Interaktionen sind darstellbar (Instanzen), aber nur wenn es ein Schema dazu gibt

- Mögliche Aussagen sind z.B.: “Alle Objekte der Art ’Student’ stehen immer mit einem Objekt der Art ’Universität’ in Beziehung” (’Student’ und ’Universität’ sind hier Schemata); “bei allen Abläufen des Verhaltens ’XY’ passiert genau folgendes: erst ’x’, dann ’y’, …”

- Nicht möglich sind Aussagen der Art: “In der Mehrheit der Fälle …”, “So ähnlich wie…”

- Ausnahmefälle entweder ausdrücklich darstellen, z.B. in einer testbaren Spezifikation, oder ignorieren, z.B. beim Entwurf (Abschätzen, ob Sachverhalt Vernachlässigbar, ob Abstraktionsniveau für die Zielsetzung des Modells passt).

- Ein UML Modell wird i.d.R. unterschiedliche Sichtweisen auf den Problemlösungsbereich (**Aspekte**) enthalten

- Aspekte erlauben eine Fokussierung, Zusammenhänge werden klarer

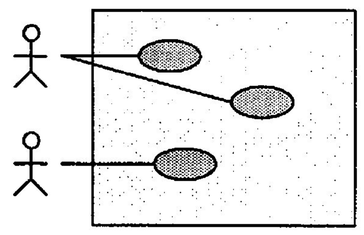
- **Statische Aspekte** beschreiben Strukturen, welche über den gesamten Lebenszyklus gelten

- **Dynamische Aspekte** beschreiben die Veränderung über die Zeit (diskrete Zeit)

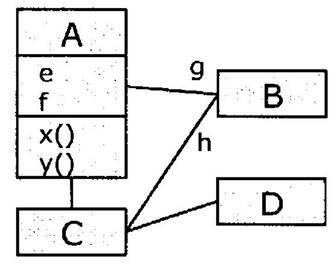
- **Softwareorganisatorische Aspekte** beschreiben die Untergliederung der Modelle selbst (in dieser Vorlesung nicht weiter behandelt)

4.3.1 Wichtige UML Schemata für statische Aspekte

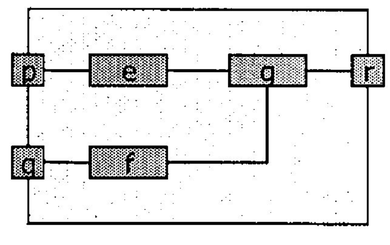
- **Anwendungsfälle**
  - die Benutzer und Nützlichkeit eines Systems
  - definiert durch Akteure und Anwendungsfälle



- **Klasse**
  - Menge gleichartiger Objekte
  - definiert durch Attribute, Operationen, und Beziehungen zu anderen Klassen

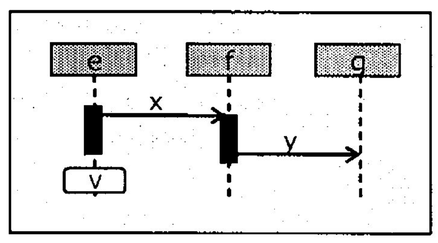


- **Kompositionsstruktur**
  - die innere Zusammensetzung eines Objekts
  - definiert durch seine Teile und die Kommunikationswege dieser untereinander bzw. mit Objekten ausserhalb

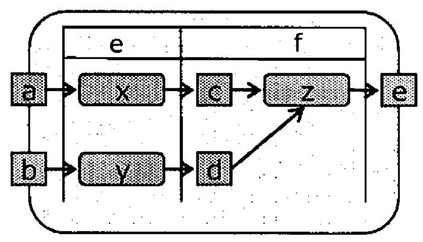


4.3.2 Wichtige UML Schemata für Dynamische Aspekte

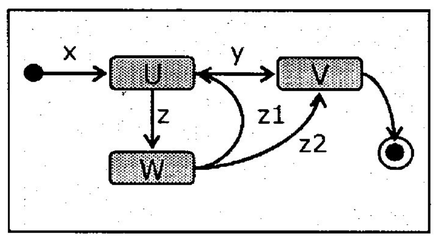
- **Interaktionen**
  - ge- bzw. verbotene Kommunikationsszenarien
  - definiert durch Folgen von Nachrichtenaustauschen zwischen Interaktionspartnern



- **Aktivität**
  - ein in sich geschlossenes, pro-aktives Verhalten
  - definiert durch Aktionen, Kontroll- und Datenfluss sowie evtl. Zuständigkeiten

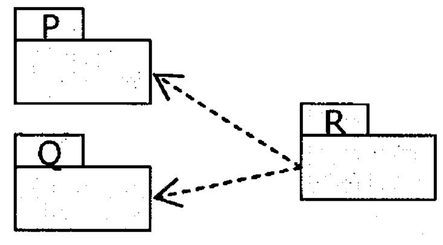


- **Zustandsautomat**
  - ein reaktives Verhalten
  - definiert durch Zustände, Ereignisse und Transitionen

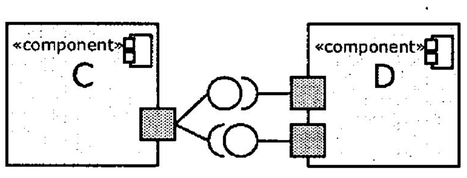


4.3.3 Wichtige UML Schemata für softwareorganisatorische Aspekte

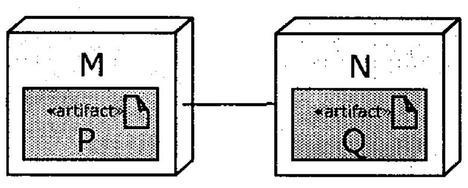
- **Pakete**
  - die Strukturierung einer Software in Verwaltungs-einheiten und deren Abhängigkeiten
  - natürlich auch für Modelle selbst



- **Komponenten**
  - zusammengesetzte Objekte, deren innerer Aufbau (Imple-mentierung) stark gekapselt ist
  - vereinigt Klassensicht und Paketsicht

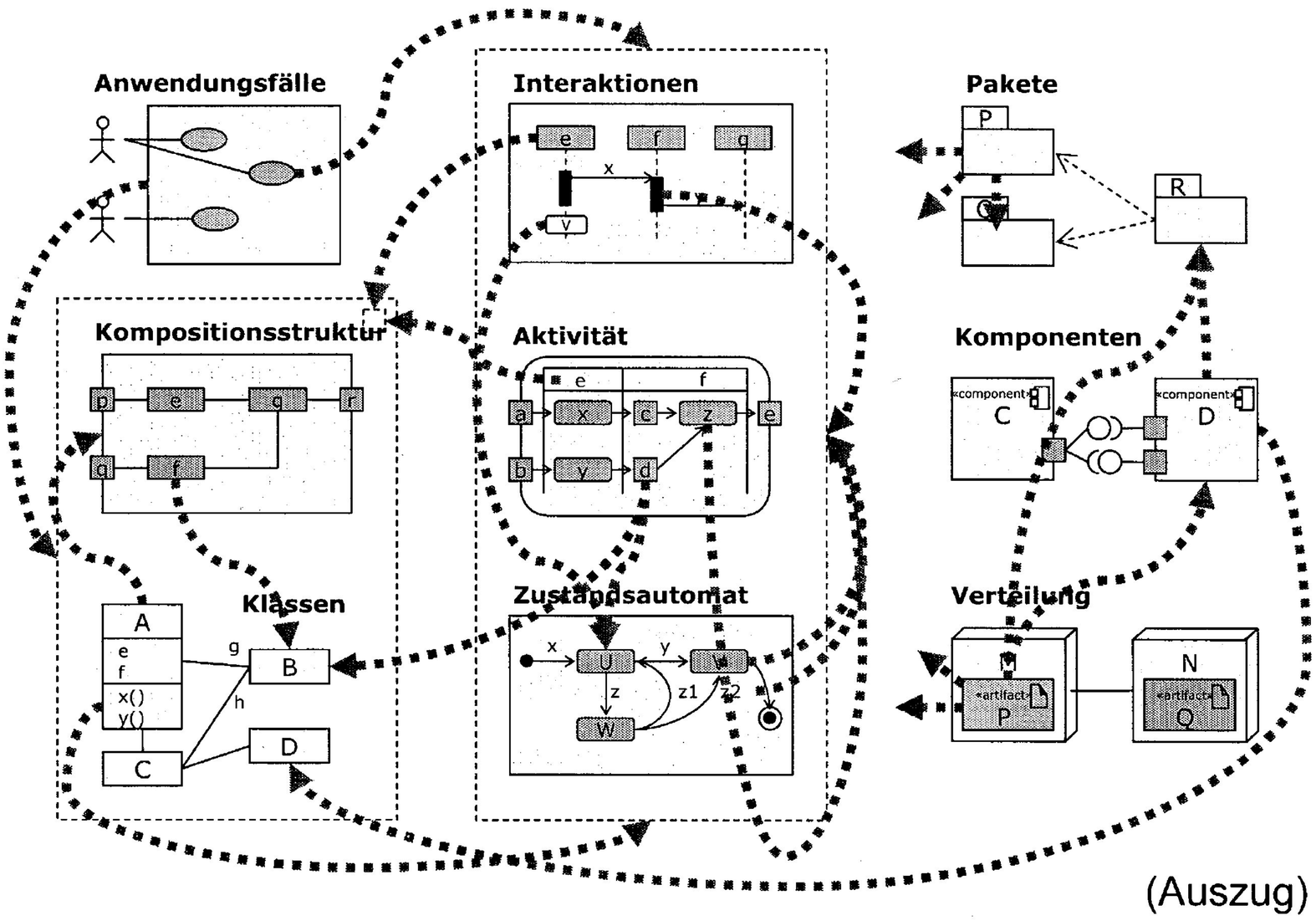


- **Verteilung**
  - Ausführungseinheiten und die Verteilung bzw. Installation von Software-Artefakten auf diesen



(Softwareorganisatorische Aspekte werden in dieser Lehrveranstaltung nicht weiter behandelt.)

4.3.4 Zusammenspiel wichtiger Schemata in UML



4.3.5 Modelle, Diagramme und Werkzeuge

- Modell vs. Diagramm
  - Ein UML-Modell ist eine Menge von Modellelementen (Klassen, Interaktionen, Zustandsmaschinen, etc.)
  - Ein UML-Diagramm ist die visuelle Darstellung von Modellelementen (eine Sicht auf ein Modell)

- Werkzeuge
  - Modellierungswerkzeuge implementieren die Sprache UML ☞ ADONIS®
    Vorteil: Vorgabe der Modellstruktur, SW-Generierungsfunktionen
    Nachteil: oft komplex
  - Zeichenwerkzeuge ermöglichen es nur, Diagramme zu zeichnen
    Vorteil: Freiheit in der Anwendung, evtl. besser tauglich für Präsentationen
    Nachteil: sehr umständlich und unzulänglich bei grösseren Modellen

- Datenaustausch
  - Standard-Speicherformat für UML-Modelle ist XMI-basiert; Versionsproblematik
  - Austausch von Diagrammen (Bildern) i.A. unproblematisch, Bilder sind jedoch nicht zum Weiterarbeiten geeignet

4.3.6 UML 2.0 Diagrammtypen

- Insgesamt 13 Diagrammtypen

- Sechs Strukturdiagramme
  - Statische Strukturen (Klassenstrukturen bis Gliederung von ganzen Systemen und Architekturen)
    - **Klassendiagramm**
    - **Objektdiagramm**
    - Komponentendiagramm
    - Kompositionsstrukturdiagramm
    - Verteilungsdiagramm
    - Paketdiagramm

- Sieben Verhaltensdiagramme
  - Abläufe zwischen statischen Teilen
    - **Aktivitätsdiagramm**
    - **Anwendungsfalldiagramm**
    - **Zustandsdiagramm**
    - **Sequenzdiagramm**
    - Interaktionsübersichtsdiagramm
    - Kommunikationsdiagramm
    - Zeitdiagramm

Literatur und Quellen:

- Martin Hitz, Gerti Kappel, Elisabeth Kapsammer, Werner Retschitzegger: UML @ Work. dpunkt Verlag 2005.
  - Die Sprache UML im Detail behandelt

- M. Blaha, J. Rumbaugh: Object-oriented Modeling and Design with UML
  - Die Methodik der Modellierung, von der Analyse bis zur Implementierung, unter Einsatz von UML

- S. Ambler: Aglie Modeling
  - Die Anwendung der Modellierung im Kontext von agiler Entwicklung; nicht UML-spezifisch

- G. Kappel: Unterlagen zu Objektorientierter Modellierung, TU Wien, 2007.

- http://www.uml.org

5 Anwendungsfalldiagramm

Inhalt

5.1 Überblick Use Case / Anwendungsfall ..... -2
5.2 Definitionen ..... -10
5.3 Akteure, Anwendungsfälle und Kommunikationsbeziehungen ..... -11
5.4 Beziehungen zwischen Anwendungsfällen ..... -15
5.5 Beziehungen zwischen Akteuren ..... -22
5.6 Beschreibung eines Anwendungsfalls ..... -24
5.7 Identifizierung von Anwendungsfällen ..... -27
5.8 Die wichtigsten Regeln zur Anwendungsfallmodellierung ..... -28

5.1 Überblick Use Case / Anwendungsfall

- Use Cases oder Anwendungsfälle sind Ausgangspunkt vieler objektorientierten Entwicklungsmethoden.

- Zusätzlich dienen sie oft auch als Basiskonzept das sich über den kompletten Analyse- und Designprozess hinweg spannt.

- Die Ausgangsfragen für den Einsatz von Anwendungsfällen sind:
  - Warum verwendet man Anwendungsfälle?
  - Wie sehen Anwendungsfälle aus?
  - Was macht man mit Anwendungsfällen, wenn man sie einmal definiert hat?

- Anwendungsfälle konzentrieren sich auf das fundamentale Problem bei der Entwicklung eines Systems: der Entwicklung einer Lösung für den Kunden bzw. Anwender, die der Kunde bzw. der Anwender auch gewünscht hat.

- Zu einem besseren Verständnis des Ansatzpunktes von Anwendungsfällen vergleichen wir, wie zwei Modellierer - einer mit Schwerpunkt Strukturierte Analyse und einer mit Schwerpunkt Objektorientierung - ein Auto sehen:

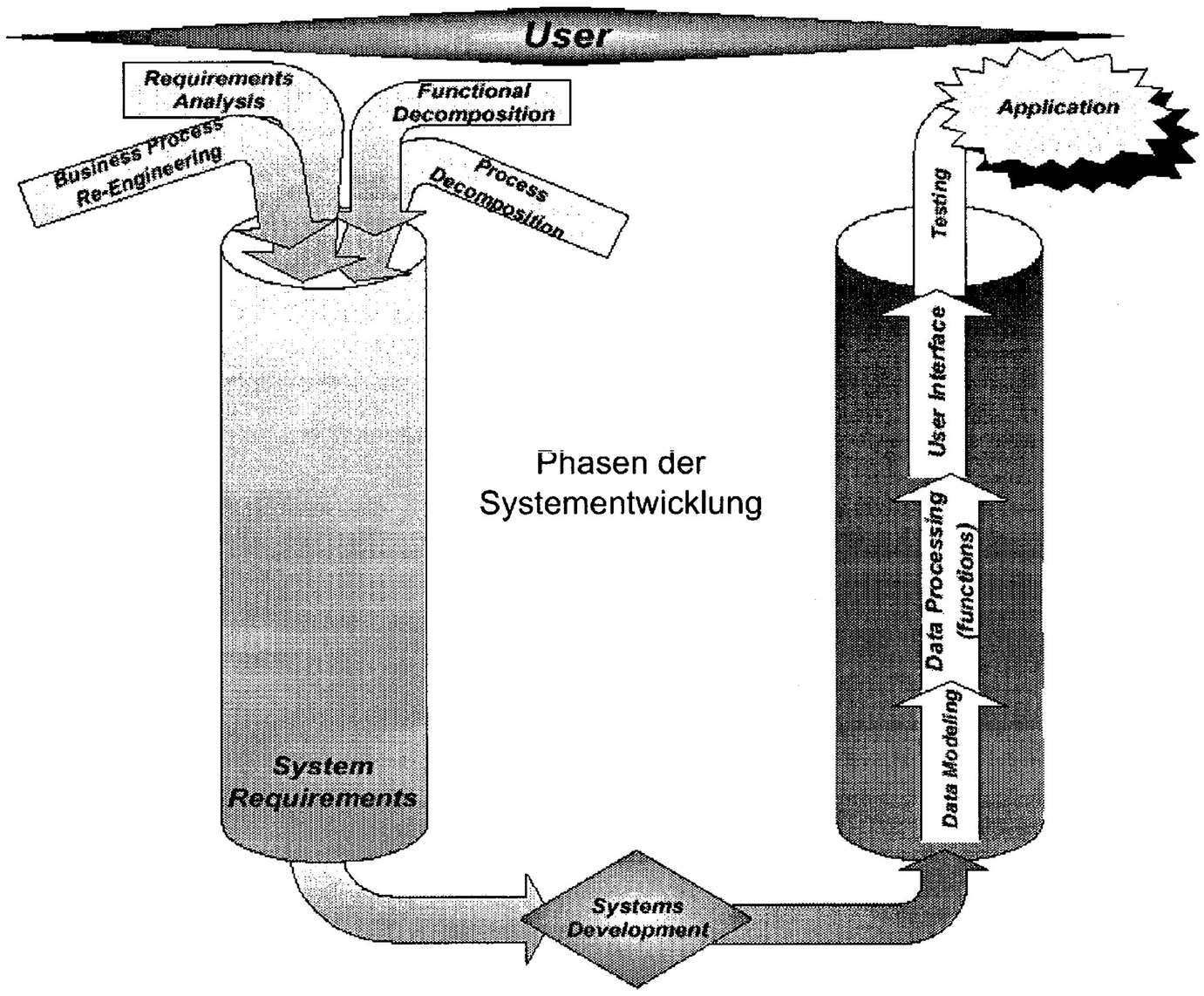
Modellierer	Beschreibung eines Autos
Strukturierte Analyse	Ein Benzintropfen gelangt vom Tank über die Benzinleitung zum Motor. Der Motor wandelt das Benzin in Energie um, wodurch sich der Motor zu bewegen beginnt. Dadurch dreht sich auch das Getriebe um ein Stück. Dies veranlasst auch eine kleine Drehung der Kurbelwelle. Das an der Kurbelwelle angebrachte Differential dreht sich somit auch ein Stück. Das Differential sitzt auf den Reifen wodurch sich auch die Reifen um ein Stück drehen. Dadurch bewegt sich das Auto ein Stück vorwärts
Objekt orientiert	Der Fahrer steigt auf das Gaspedal, wodurch sich das Auto fortbewegt. Der Motor stoppt sobald der Fahrer die Zündung ausschaltet.

- Somit ergibt sich auch ein unterschiedlicher Ansatz in der Systementwicklung zwischen strukturierter Analyse und einem Anwendungsfall-zentriertem Ansatz.

- Dabei können wir die Entwicklung eines Systems in zwei Phasen beschreiben.
  - Die erste Phase dient zur Beschreibung der fokussierten Geschäftsumgebung
  - Die zweite Phase dient zur Umsetzung dieser Geschäftsumgebung in einem System bzw. in einer Anwendung.

- In der ersten Phase erfolgt
  - die Identifizierung des Problems
  - analytisch ausgerichtete Erfassung der Systemanforderungen mittels
    - Process Decomposition
    - Functional Decomposition
    - Business Flow-Charts
    - Workflow-Diagrammen
    - Business Process Reengineering Aktivitäten
    - etc.

- Am Ende der ersten Phase entsteht ein guter Überblick über die Geschäftsumgebung. Es entstehen Ansatzpunkte zur Schaffung eines Systems.



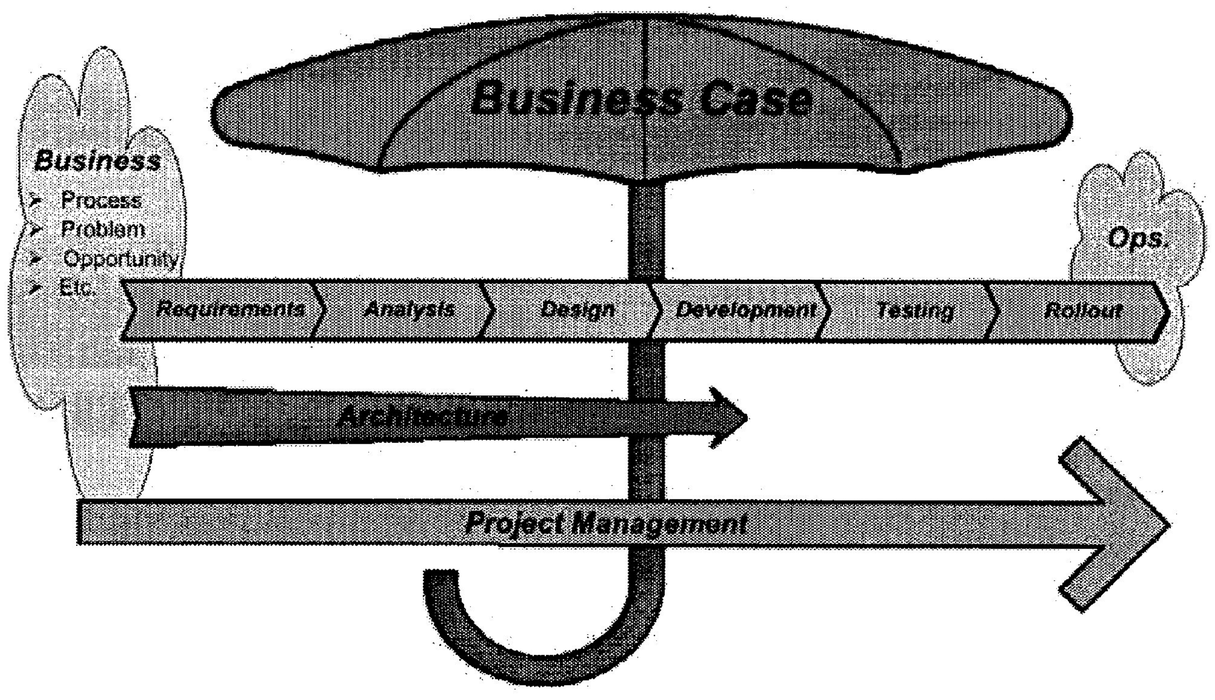
- Während der ersten Phase ist der Anwender ständig eingebunden. Der Anwender treibt dabei den Prozess.

- Auf Basis der Erkenntnisse der ersten Phase gestaltet sich die zweiten Phase:
  - Zuerst erfolgt die Modellierung von einer datenorientierten Sichtweise: Es entstehen ER-Diagramme und Datenflussdiagramme.
  - Auf Basis der datenorientierten Diagramme werden Diagramme erstellt, die festhalten wie die Daten manipuliert werden.
  - Danach muss auf diese Diagramme noch eine entsprechendes User Interface aufgesetzt werden.
  - Letztendlich entsteht am Ende der zweiten Phase das Anwendungssystem.

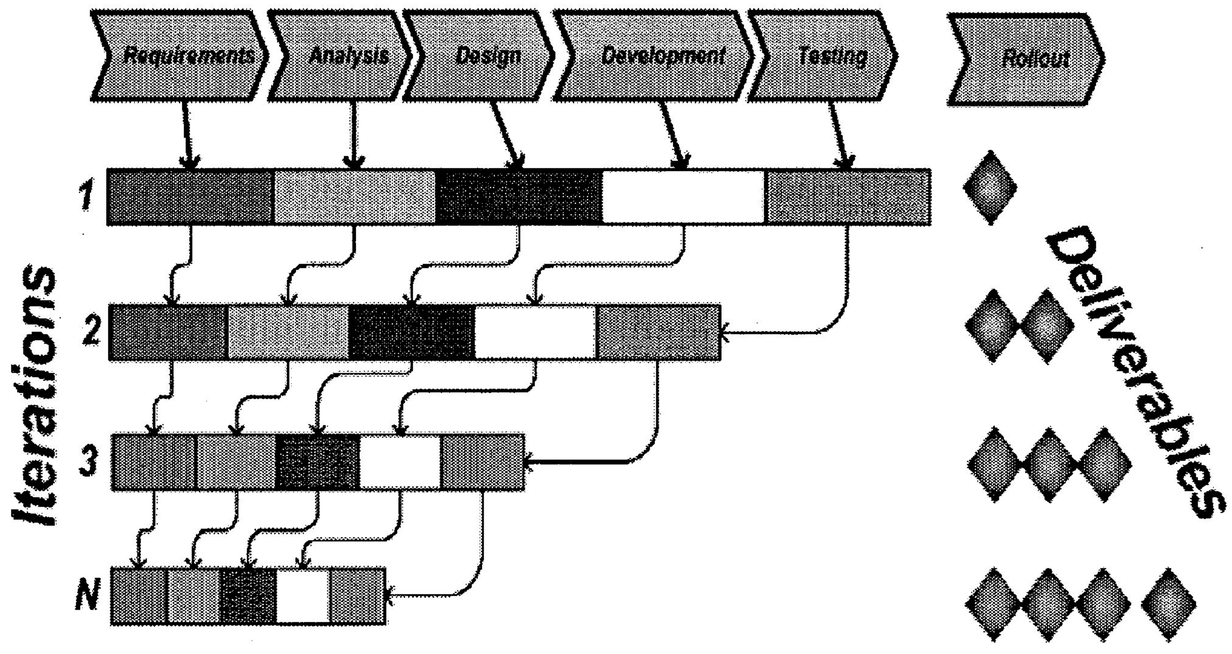
- Mit den Prozessen in der zweiten Phase ist vorwiegend die IT-Abteilung befasst. Dadurch ergeben sich folgende Probleme:
  - Die IT-Abteilung interpretiert die Anwenderanforderungen aus ihrer eigenen, meist rein technischen Sichtweise
  - Dadurch erfolgt die Systementwicklung vorallem von einer IT-Perspektive und nicht von einer Anwender-Perspektive
  - Das Risiko ein dem Anwenderwunsch entsprechendes System zu schaffen ist hoch.
  - Weiters können durch eine scharfe Trennung der beiden Phasen neue Erkenntnisse und Änderungen in der Geschäftsumgebung nicht in das sich entwickelnde System einfließen

- Es bedarf einer Entwicklungsmethode in welcher der Anwender bis zum Ende involviert bleibt und die flexibel genug ist um auf Umgebungsänderungen zu reagieren.

- Die Verwendung einer Anwendungsfall-zentrierten Methode erfüllt diese Anforderungen. Sämtliche entstehende Dokumente der Systemdokumentation müssen immer gegen den entsprechenden Anwendungsfall evaluiert werden.



- Eine Anwendungsfall-zentrierte Methode ist eine inkrementelle und iterative Entwicklungsmethode.



5.2 Definitionen

- Das Konzept der Anwendungsfälle stammt von Ivar Jacobson's "Object-Oriented Software Engineering" (OOSE). Ein Anwendungsfall ist dort definiert als eine Sequenz von Transaktionen innerhalb eines Systems, deren Aufgabe es ist einen für den einzelnen Akteur (Anwender) identifizierbaren Nutzen zu erzeugen.

- Ein Akteur ist eine Rolle, die jemand oder etwas einnimmt und die in Beziehung zum Geschäftsbereich steht. Oder alternativ:
  - Alles, das Informationen mit dem System austauschen muss
  - Alles, das mit dem System interagiert

- Transaktionen innerhalb eines Systems implizieren, dass den Akteuren eine Reihe von Möglichkeiten geboten werden um mit dem System zu kommunizieren und dass durch sie ein messbarer Nutzen erzeugt wird.

- Ein messbarer Nutzen impliziert, dass die Ausführung einer Transaktion eine sichtbare, quantifizierbare und/oder qualifizierbare Auswirkung auf jene Dinge hat, die außerhalb des Systems liegen, im speziellen auf den Akteur.

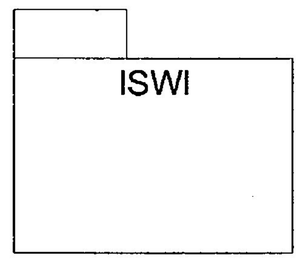
- Eine Transaktion ist definiert als eine Menge von untrennbaren Aktivitäten, die entweder alle zusammen oder gar nicht ausgeführt werden.

5.3 Akteure, Anwendungsfälle und Kommunikationsbeziehungen

- Ein System kann durch seine zur Verfügung zu stellenden Anwendungsfälle und durch seine Grenzen beschrieben werden.

- Ein System wird durch ein Rechteck dargestellt, das die Anwendungsfälle enthält.

- Der Name des Systems wird innerhalb des Rechtecks angegeben, die Kanten des Rechtecks symbolisieren die Systemgrenzen



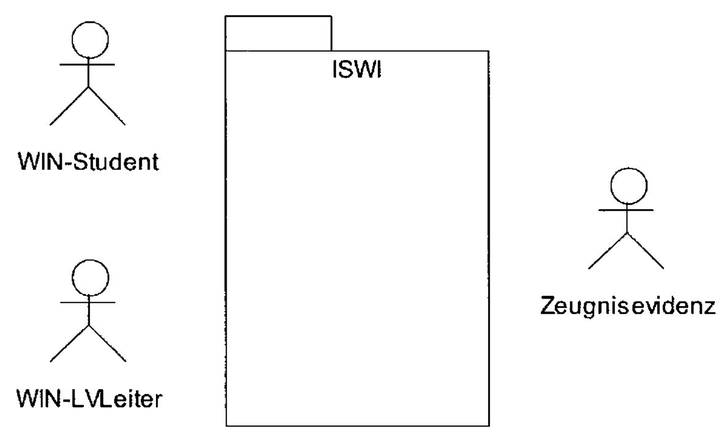
- Durch Akteure wird festgelegt wer mit dem System interagieren soll.

- Akteure stehen klar außerhalb des Systems. Sie benutzen das System, indem sie die Ausführung von Anwendungsfällen initiieren, oder sie werden vom System benutzt, indem sie selbst Funktionalität zur Realisierung einzelner Anwendungsfälle zur Verfügung stellen.

- Obwohl es für das Auffinden von Akteuren hilfreich ist, von konkreten Individuen, die mit dem System arbeiten, auszugehen, repräsentieren Akteure **keine** Objekte, sondern *Rollen*, die von Objekten gespielt werden können. Jeder Akteur hat einen Namen, der seiner Rolle entspricht.

- Akteure werden durch ein "Strichmännchen" dargestellt. Der Name des Akteurs wird unterhalb der Figur angegeben.

- Ein Anwendungsfall beschreibt ein bestimmtes Verhalten, das von einem zu entwickelnden System erwartet wird.



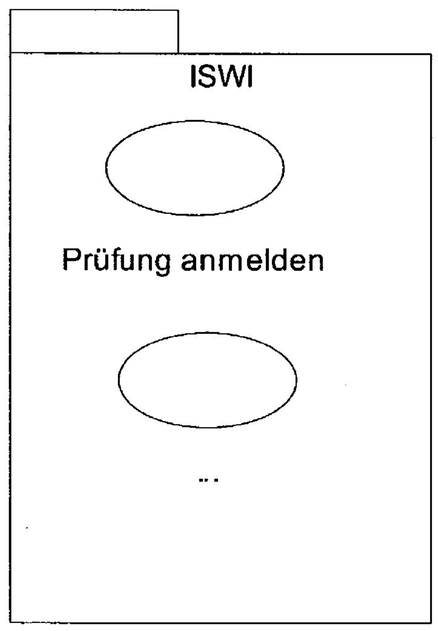
- Die Summe aller Anwendungsfälle zusammen macht die Systemfunktionalität des Gesamtsystems aus.

- Die Schwierigkeit in der Praxis ist es oft sämtliche Anwendungsfälle vollständig zu dokumentieren.

- Ein Anwendungsfall wird durch eine Ellipse dargestellt. Der Name des Anwendungsfalls wird innerhalb oder unterhalb der Ellipse angegeben.

- Die bloße “Ellipse” ist wertlos, wenn nicht die Semantik jedes Anwendungsfalles (in Prosa) festgehalten wird.

- Die identifizierten Anwendungsfälle, das sie umschließende System und die kommunizierenden Akteure werden in einem Anwendungsfalldiagramm zusammengefaßt.

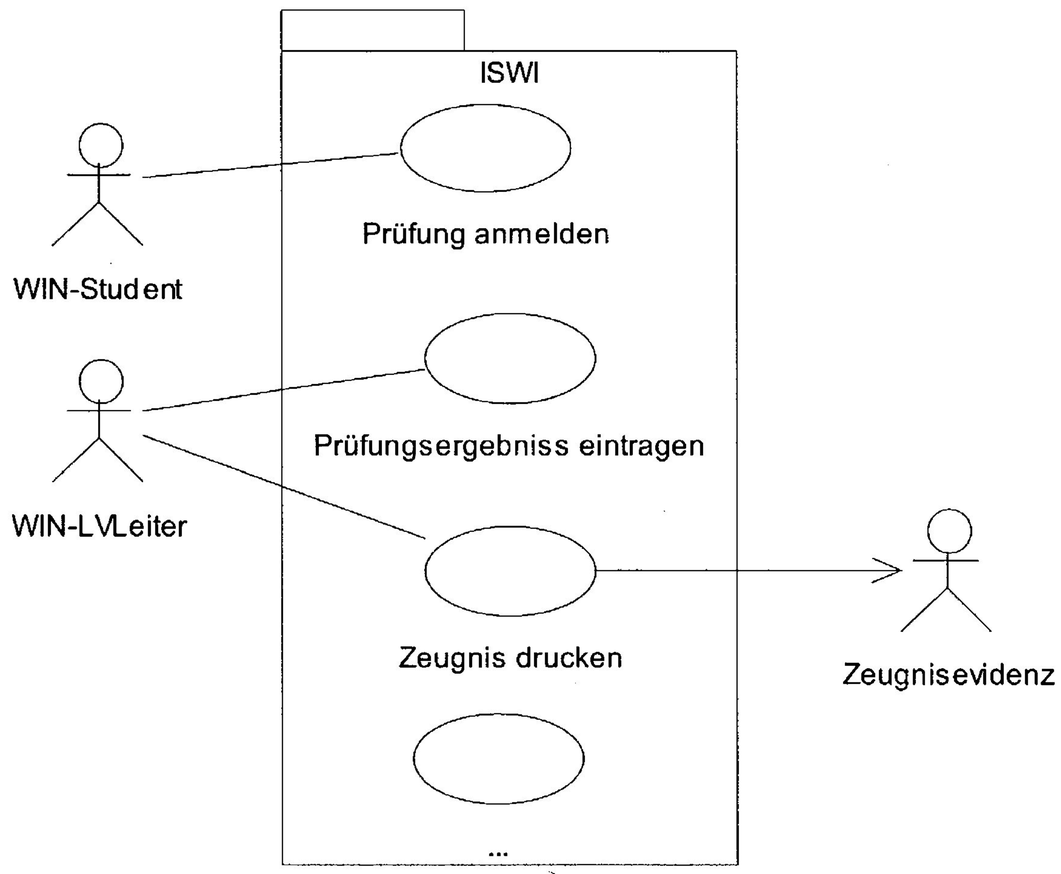


- Jeder Akteur muss zu mindestens einem Anwendungsfall eine Kommunikationsbeziehung haben. Er kann aber auch mit mehreren Anwendungsfällen kommunizieren.

- Eine Kommunikationsbeziehung wird zwischen genau einem Akteur und genau einem Anwendungsfall spezifiziert.

- Kommunikationsbeziehungen sind normalerweise ungerichtet, d.h. die Kommunikation kann in beide Richtungen erfolgen.

- Wenn auch nicht in UML festgelegt, wird öfters auch eine gerichtete Kommunikationsbeziehung verwendet, falls die Kommunikation nur in eine Richtung erfolgt.



5.4 Beziehungen zwischen Anwendungsfällen

- Anwendungsfälle können nicht nur eine Kommunikationsbeziehung zu Akteuren haben, sondern können auch untereinander in Beziehung stehen.

- Es werden 3 Arten von Beziehungen zwischen Anwendungsfällen unterschieden:
  - Include-Beziehung
  - Extend-Beziehung
  - Generalisierungsbeziehung

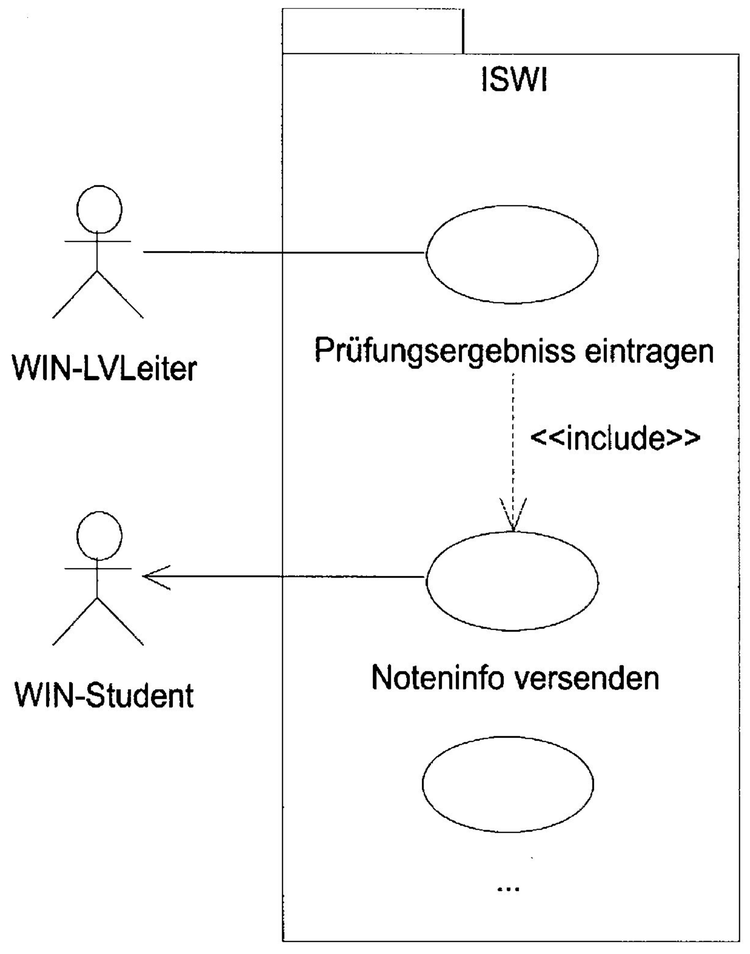
Include-Beziehung

- Die Include-Beziehung ist eine gerichtete Beziehung zwischen zwei Anwendungsfällen.

- Eine Include-Beziehung von Anwendungsfall A nach Anwendungsfall B besagt, dass das Verhalten von B in A eingefügt *wird*.

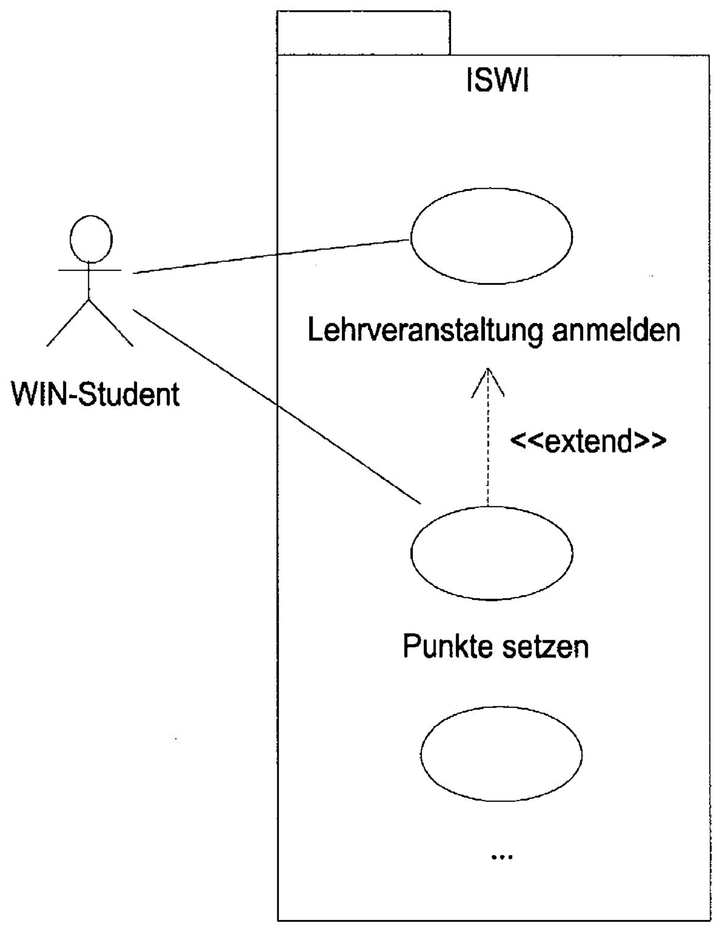
- Die Include-Beziehung von A nach B besagt, dass jedesmal wenn A ausgeführt wird auch der Anwendungsfall B ausgeführt wird.

- Die Include-Beziehung wird durch einen Abhängigkeitspfeil dargestellt, der mit dem Schlüsselwort <<include>> beschriftet ist.

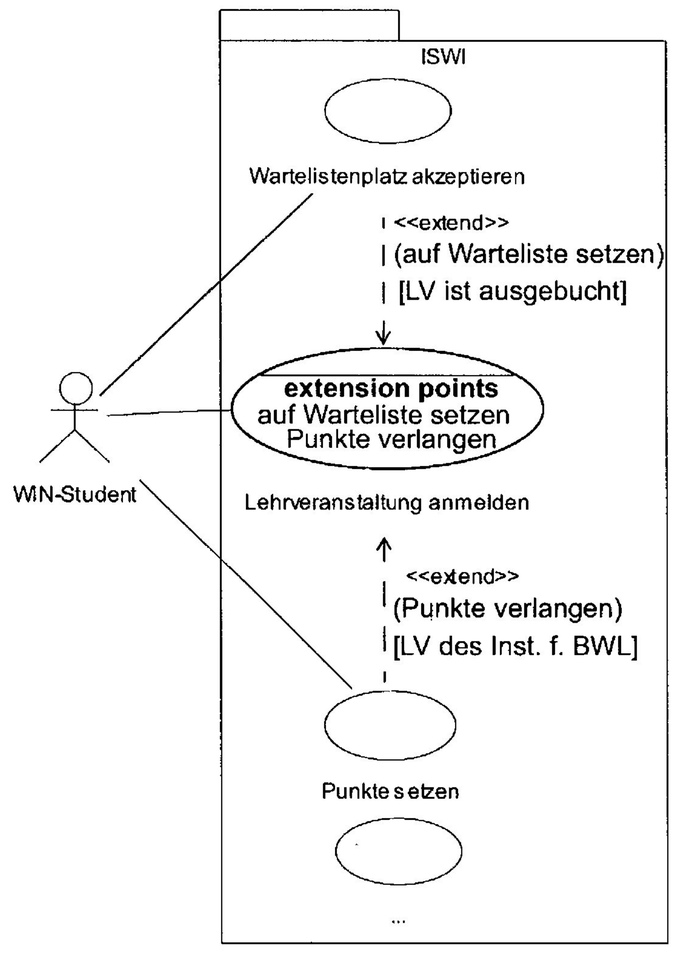


Extend-Beziehung

- Die Extend-Beziehung ist ebenfalls eine gerichtete Beziehung zwischen zwei Anwendungsfällen.
- Eine Extend-Beziehung von Anwendungsfall B nach Anwendungsfall A besagt, dass das Verhalten von B in A eingefügt werden *kann*.
- Die Extend-Beziehung wird ebenfalls durch einen Abhängigkeitspfeil dargestellt, der mit dem Schlüsselwort <<extend>> beschriftet ist.
- Für jede Extend-Beziehung können eine oder mehrere Erweiterungsstellen im zu erweiternden Anwendungsfall definiert werden, die angeben, wo der zu erweiternde Anwendungsfall einzufügen ist.



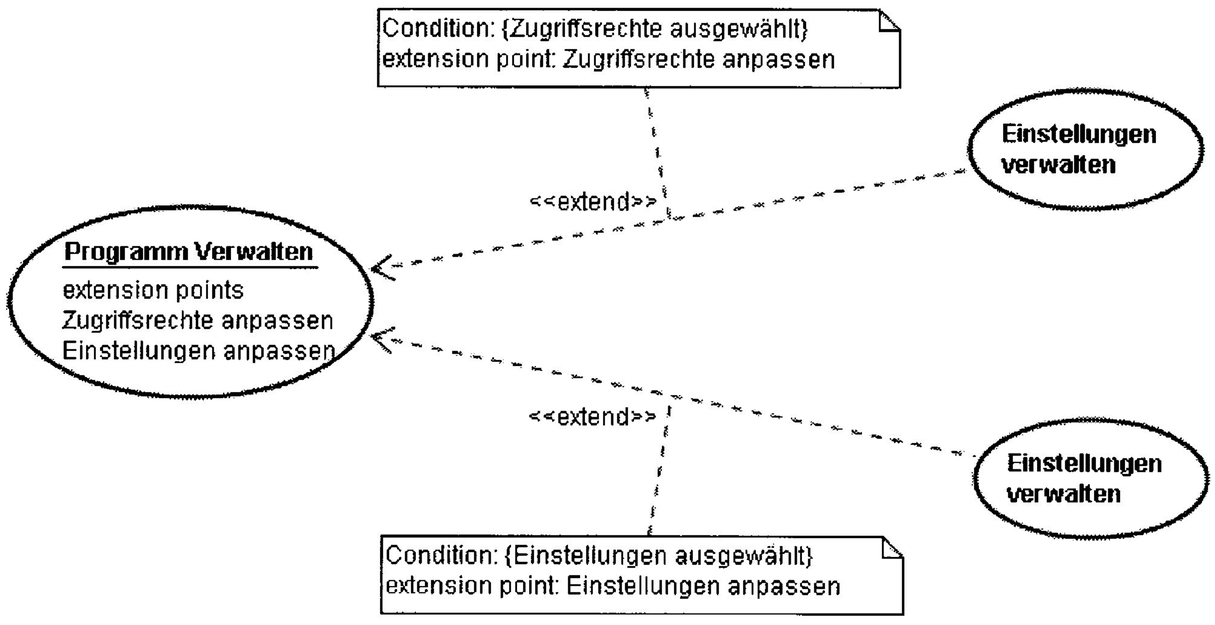
- Die Erweiterungsschnittstellen, die sogenannten Extension Points, sind innerhalb des Anwendungsfalles A eindeutig identifiziert und können in diesem in einem eigenen Abschnitt des Anwendungsfallsymbols aufgelistet werden.
- Die Erweiterungsschnittstellen müssen nicht unbedingt mit den Namen der zu erweiternden Anwendungsfälle übereinstimmen.
- Werden mehrere Erweiterungsschnittstellen spezifiziert, muss eine Zuordnung zwischen dem zu erweiternden Anwendungsfall und einer Erweiterungsstelle, wo er eingefügt werden soll, vorgenommen werden. Dies geschieht durch Angabe der entsprechenden Extend-Beziehung, indem der Name der Erweiterungsstelle in runden Klammern unter das <<extend>>-Schlüsselwort gesetzt wird.



- Da die Extend-Beziehung eine optionale Beziehung zum Ausdruck bringt, muss für jeden zu erweiternden Anwendungsfall auch eine Bedingung spezifiziert werden, die erfüllt sein muss, damit der Anwendungsfall eingefügt werden kann.

- Die Bedingung wird in eckigen Klammern ebenfalls zur Extend-Beziehung vermerkt. Die Spezifikation einer entsprechenden Einschränkung in einer Notiz und die Annotation dieser Notiz zur dazugehörigen Extend-Beziehung ist eine weitere Darstellungsmöglichkeit.

- UML 2.0 Notation:

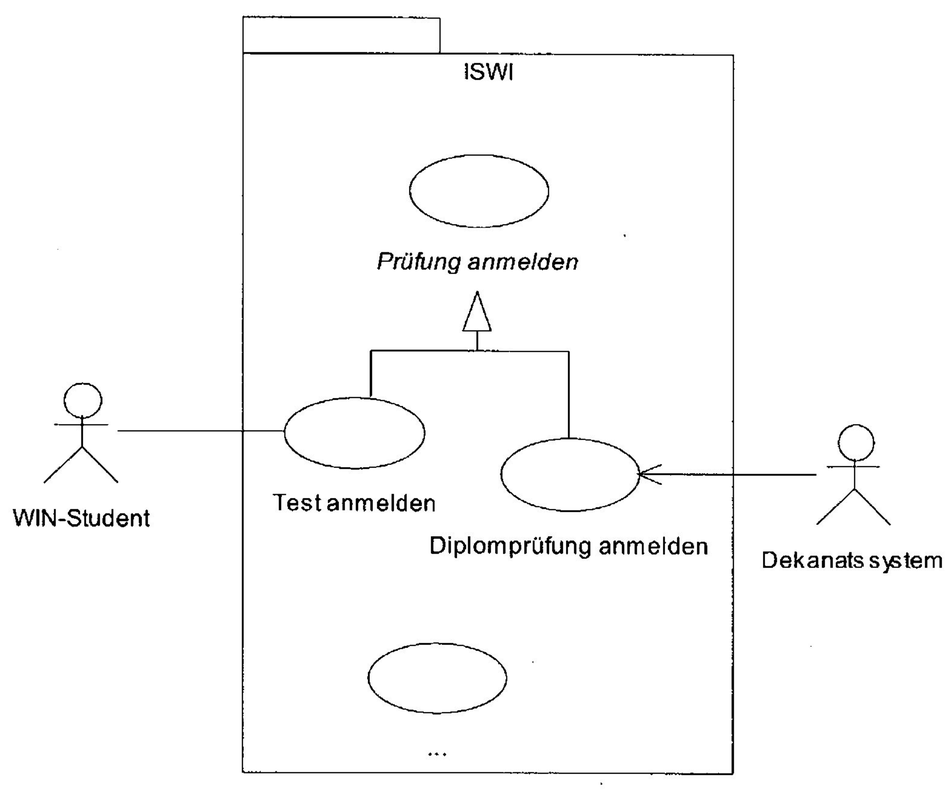


Generalisierungsbeziehung

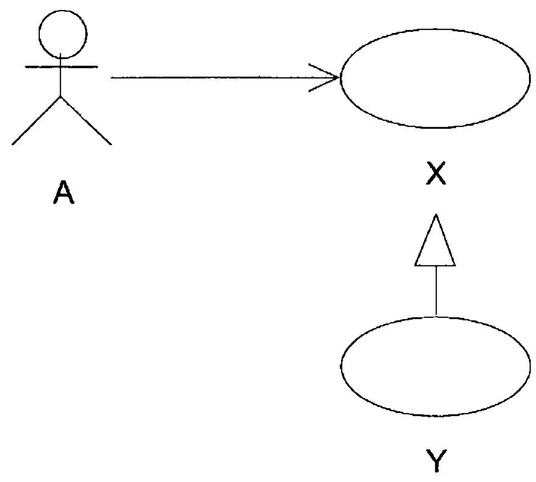
- Eine Generalisierungsbeziehung zwischen zwei Anwendungsfällen hat eine zur Generalisierungsbeziehung zwischen Klassen äquivalente Semantik.

- Der erbende Anwendungsfall erbt das gesamte Verhalten des vererbenden Anwendungsfalls und kann dieses überschreiben oder ergänzen.

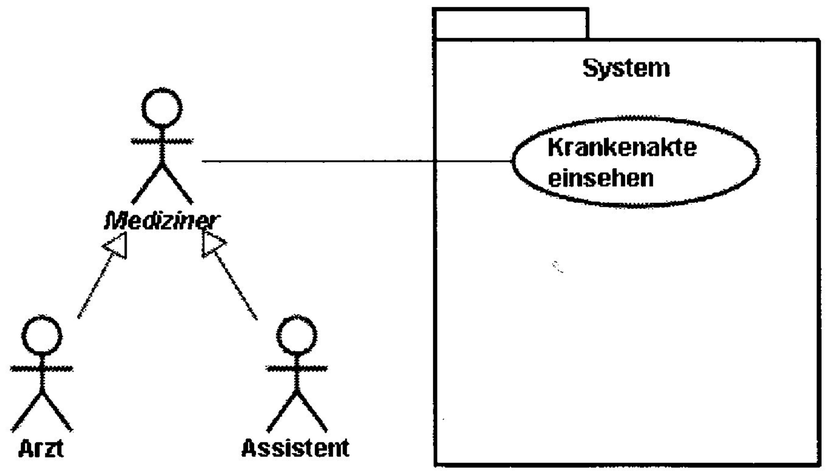
- Die Darstellung der Generalisierungsbeziehung ist ebenfalls identisch zu der zwischen Klassen.



- Ein interessanter Aspekt bei der Generalisierungsbeziehung zwischen Anwendungsfällen ist die Vererbung der Kommunikationsbeziehung. Damit erbt der spezialisierte Anwendungsfall auch die Kommunikationsbeziehungen des generellen Anwendungsfalls. Somit müssen die Kommunikationsbeziehungen zum Akteur im spezialisierten Anwendungsfall nicht noch einmal eingezeichnet werden.



- Generalisierungsbeziehungen können auch zwischen Akteuren bestehen (vgl. Folie 7-23).



5.5 Beziehungen zwischen Akteuren

- Zwischen Akteuren gibt es als einzige Beziehungsart die Generalisierungsbeziehung.

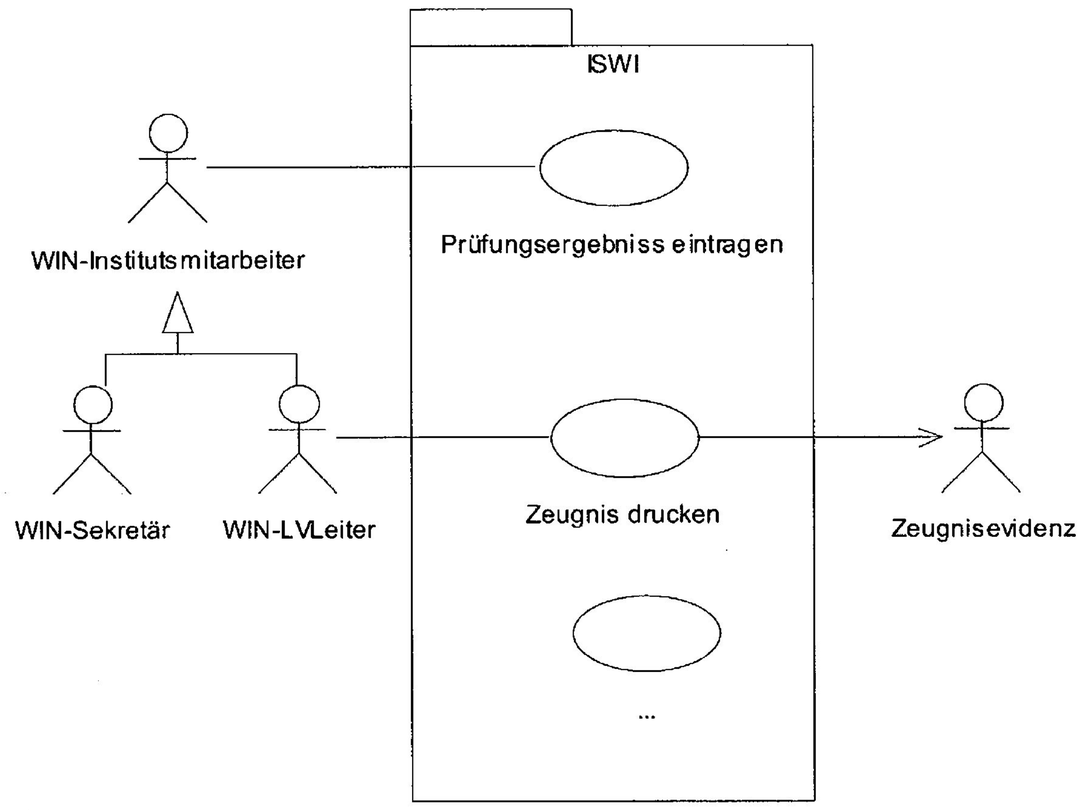
- Eine Generalisierungsbeziehung zwischen dem generellen Akteur A und dem speziellen Akteur B besagt, dass B mit denselben Anwendungsfällen wie A kommunizieren kann.

- Ein Akteur kann von mehreren Akteuren erben und kann selbst mehrfach beerbt werden.

- Die Generalisierung von Akteuren kann auch benutzt werden, um zu unterscheiden, ob mehrere Akteure gemeinsam mit einem Anwendungsfall kommunizieren müssen oder ob mehrere Akteure mit demselben Anwendungsfall kommunizieren können.

- Sollten mehrere Akteure gemeinsam mit einem Anwendungsfall kommunizieren, so wird direkt eine Kommunikationsbeziehung zwischen jedem Akteur und dem Anwendungsfall definiert.

- Sollte nur einer von mehreren Akteuren mit einem Anwendungsfall kommunizieren müssen, so können die einzelnen Akteure zu einem (eventuell abstrakten) generellen Akteur zusammengefasst werden, der mit dem Anwendungsfall kommuniziert.



5.6 Beschreibung eines Anwendungsfalls

- Die Abbildung eines Anwendungsfalls in einem Anwendungsfalldiagramm ist wertlos, wenn nicht klar ist, welche Transaktionen im entsprechenden Anwendungsfall ausgeführt werden sollen.

- Um eine konsistente Beschreibung von mehreren Anwendungsfällen durch verschiedene Personen zu garantieren eignet es sich in einem Projekt ein Template für die Beschreibung zu verwenden.

- Das Template soll dazu dienen eine semiformale Beschreibung der Anforderungen unter konsistenter Verwendung einer Prosa zu garantieren.

- Die Beschreibungen innerhalb eines Templates sollen NICHT auf Implementierungsdetails Rücksicht nehmen. Bemerkungen zur Implementierung sollten getrennt dokumentiert werden.

- Das nachfolgende Schema stellt einen Vorschlag für ein Template dar. Es kann aber kein semiformales, allumfassendes Template zur Erfassung aller möglicher Strukturen, die in irgendeinem Anwendungsfall vorkommen können, geben.

Zweiter Abschnitt

- Im zweiten Abschnitt werden die **Attribute** der Klasse aufgelistet. Die einfachste Form ist die Angabe des Attributnamens.

- Die **Sichtbarkeitsangabe** ist C++ nachempfunden:
  - public (+) Attribute stehen für öffentliche bzw. uneingeschränkte Sichtbarkeit.
  - private (-) Attribute sind nur innerhalb der Klasse sichtbar und können nur in Methoden der Klasse verwendet werden.
  - protected (#) Attribute signalisieren Sichtbarkeit für die entsprechende Klasse und deren Unterklassen.
  - Sprachspezifische Erweiterungen sind möglich.

- Die **Typangabe** ist optional und kann sprachspezifisch erfolgen.

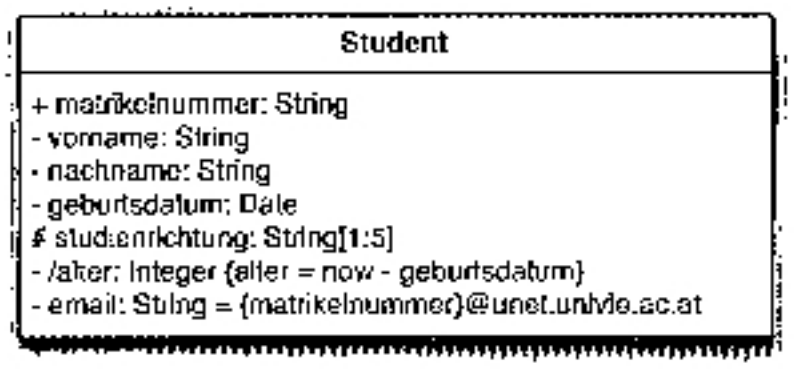
- Einem Attribut kann ein **Initialwert** zugewiesen werden.
  Dabei kann eine beliebige sprachspezifische Syntax gewählt werden.

- Es ist auch eine **Multiplizitätsangabe** für Attribute möglich.
  Die Anzahl der möglichen Wiederholungen wird als Intervall ausgedrückt, z.B. [1..*]

- Die Tatsache, dass ein **Attribut abgeleitet** ist, d.h., dass seine Ausprägungen aus anderen Informationen berechenbar sind, lässt sich durch einen dem Namen vorangestellten Schrägstrich festhalten.

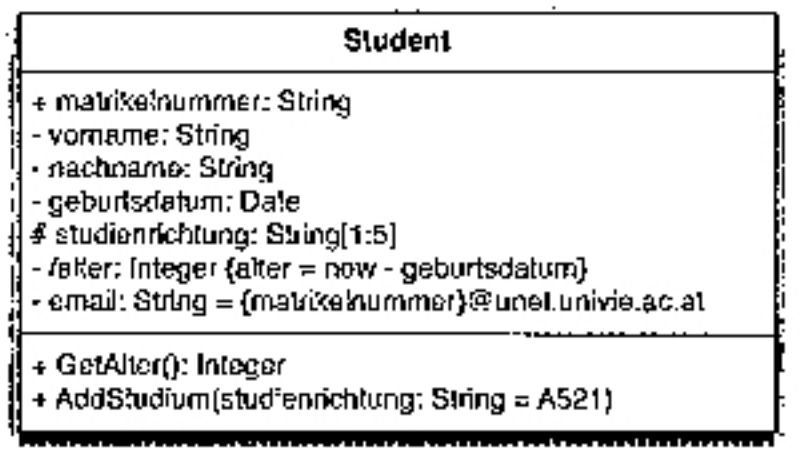
  Zusätzlich kann die Berechnungsvorschrift als Einschränkung in geschwungenen Klammern angegeben werden.

- Graphische Darstellung:



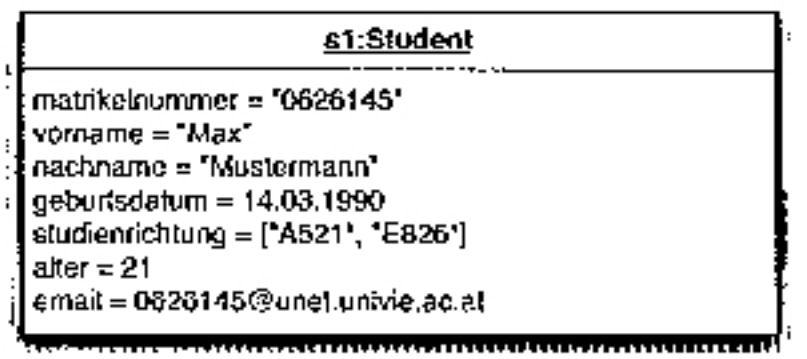
Dritter Abschnitt

- Im dritten Abschnitt werden die von der Klasse angebotenen **Operationen** angeführt.
- Für jede Operation wird die Parameterliste angegeben. Wobei für jeden Parameter zumindest der Name angegeben werden muss. Typangabe, Datenflussrichtung und Standardwert sind für die Parameter optional.
- Nach der Parameterliste kann der Ergebnistyp angegeben werden. Wird er ausgelassen, so bedeutet dies, dass die Operation nichts zurückliefert.
- Die Angabe der Sichtbarkeit (public, private, protected) ist entsprechend jener für Attribute.
- Weitere Aspekte einer Operation können wiederum in einer Eigenschaftsliste am Ende der Spezifikation angegeben werden.



6.1.2 Objekt

- UML benützt einen einheitlichen Mechanismus um generelle **Instanzen** von dem dazu gehörigen Typ zu unterscheiden, wie etwa Objekte von Klassen, in dem die Instanz dasselbe graphische Symbol wie der Typ verwendet, jedoch der Name (und falls vorhanden die Typangabe) unterstrichen wird.
- Objekten, als Instanzen von Klassen, können für ihre Attribute Werte zugeordnet werden.
- Bei der Angabe des Objektes kann entweder die Angabe des Typs oder des Objektnamens (anonyme Objekte) entfallen
- Graphische Notation:

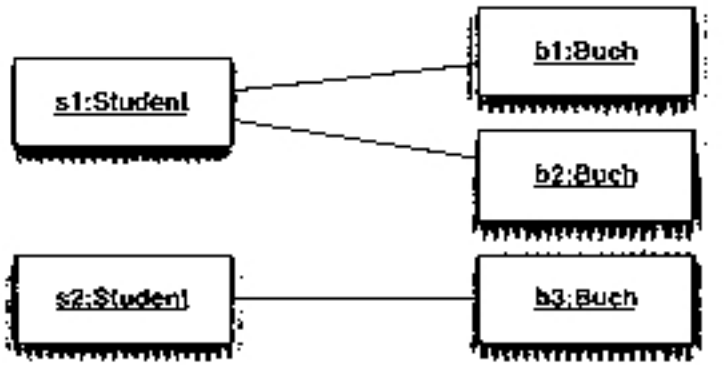


6.1.3 Allgemeine Assoziation

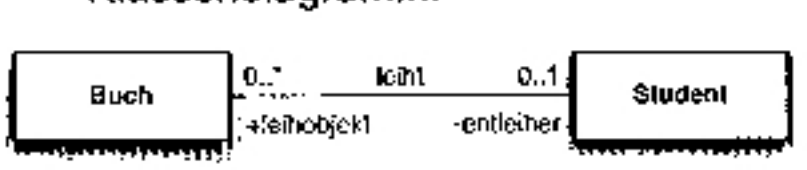
- Allgemeine Assoziationen beschreiben die gemeinsame Struktur einer Menge von in der Regel statischen Beziehungen zwischen Objekten.

- Binäre Assoziationen, die Beziehungen zwischen je zwei Klassen beschreiben, werden in UML durch einfache Verbindungslinien (Kanten) zwischen beteiligten Klassen dargestellt.

- Graphische Darstellung im Objektdiagramm:



- Graphische Darstellung im Klassendiagramm:



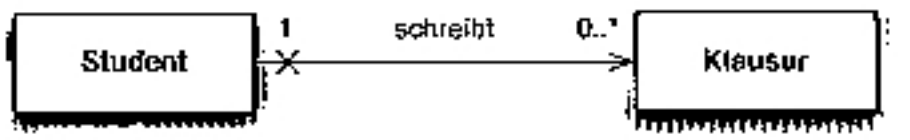
- Um die Aussagekraft von Assoziationen zu erhöhen können weitere Zusatzaussagen getroffen werden: Assoziationsname, Rollenbezeichnungen, Multiplizitäten.

- Der Name einer Assoziation wird in der Regel in der Mitte der Assoziationskante notiert. Wie bei abgeleiteten Attributen können abgeleitete Assoziationen durch einen vorangestellten Schrägstrich ausgezeichnet werden.

- Assoziationsrollen geben an, welche Rollen die beteiligten Objekte im Rahmen der Beziehung spielen. Angaben über die Rollen werden an den Enden der Verbindungslinien notiert, sie werden daher auch manchmal als Assoziationsenden bezeichnet.

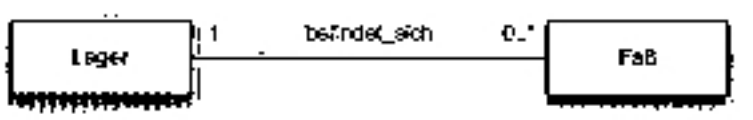
- Ein bestimmtes Objekt kann mit einer gewissen Anzahl von Partnerobjekten der gegenüberliegenden Klasse verbunden sein. Dies lässt sich aus einem geeigneten Objektdiagramm erkennen. Im Klassendiagramm lässt sich natürlich nur die mögliche Anzahl in Form der Multiplizität der Rolle festhalten. Die Multiplizität kann in Form eines Intervalls angegeben werden.

- Assoziationskanten können auch gerichtet sein, um explizit anzugeben, in welche Richtung die Navigation von einem Objekt zu seinem Partnerobjekt erfolgen kann.

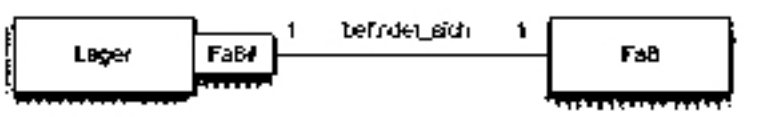


6.1.4 Qualifizierte Assoziation

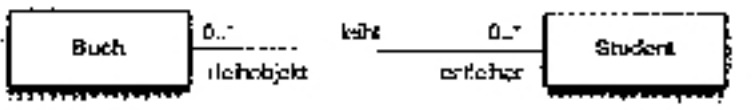
- In manchen Fällen ist es sinnvoll, bei Assoziationsrollen mit Multiplizität größer als eins die Menge der Objektbeziehungen durch ein oder mehrere sogenannte qualifizierende Attribute in disjunkte Teilmengen zu zerlegen.



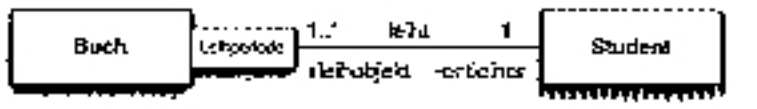
- Handelt es sich bei den qualifizierenden Attributen um einen Schlüssel der Assoziation so entstehen einelementige Teilmengen, und die Multiplizität der Assoziation wird auf eins reduziert.



- Qualifizierende Attribute werden in einem kleineren Rechteck an jenem Ende der Assoziation notiert, dessen Rolle den Kontext für die Qualifikation darstellt.



- Weiteres Beispiel bei historischen Daten:



6.1.5 Attributierte Assoziation (Assoziationsklasse)

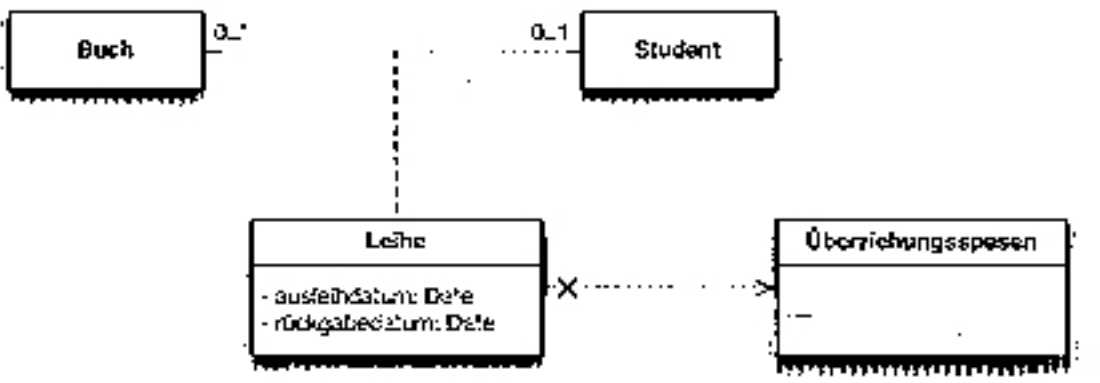
- Wenn Assoziationen durch eigene (nicht qualifizierende) Attribute näher beschrieben werden sollen oder ihrerseits an anderen Assoziationen teilnehmen sollen, müssen sie zuerst "klassenwertig" gemacht werden.

- Sogenannte Assoziationsklassen verwenden das Klassensymbol, das durch eine strichlierte Linie mit der Assoziationskante verbunden ist.

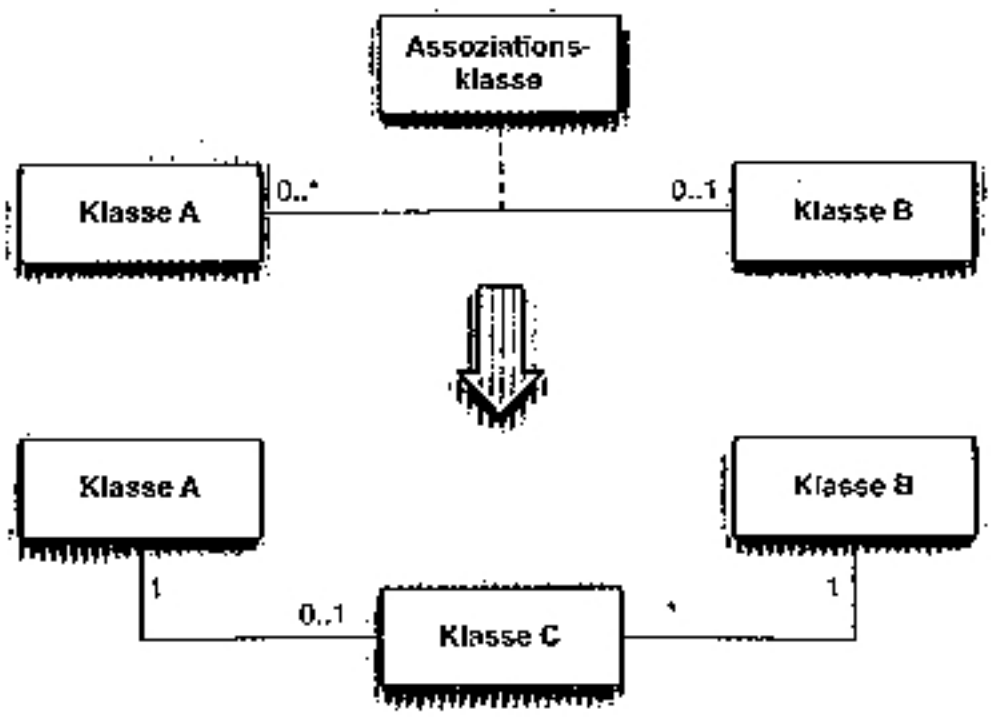
- Semantisch sind Assoziation und Assoziationsklasse identisch, d.h. der Name der Assoziationsklasse entspricht dem Namen der Assoziation.

- Graphische Notation:

- Die attributierte Assoziation sollte in der Praxis vermieden werden, da sie nicht objektorientiert ist

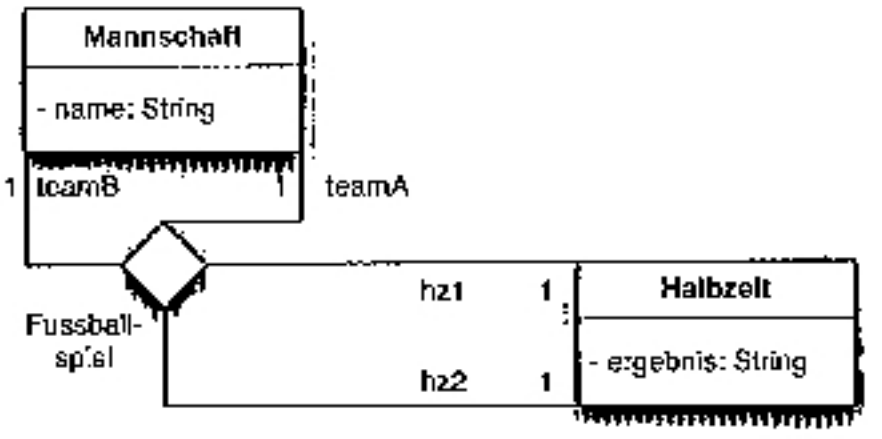


- Eine bidirektionale Assoziation ist als Kurzschreibweise für zwei einzelne entgegengesetzt gerichtete Assoziationen zu sehen



6.1.6 Mehrgliedrige Assoziation

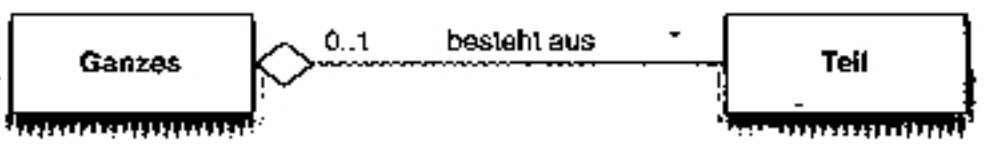
- In manchen Fällen kann eine Beziehung mehr als zwei Partnerobjekte betreffen.
- Sie wird - wie bei ER-Diagrammen - durch eine Raute, die mit den teilnehmenden Klassen verbunden ist, dargestellt.



- Die mehrgliedrige Assoziation ist mit verschiedenen semantischen Detailproblemen (Multiplizitätszuordnung, Verantwortlichkeiten) behaftet, ähnlich wie die attributierte Assoziation kein objektorientiertes Konzept und inder Praxis fehleranfällig. Man sollte daher den darzustellenden Sachverhalt mit gewöhnlichen Assoziationen beschreiben

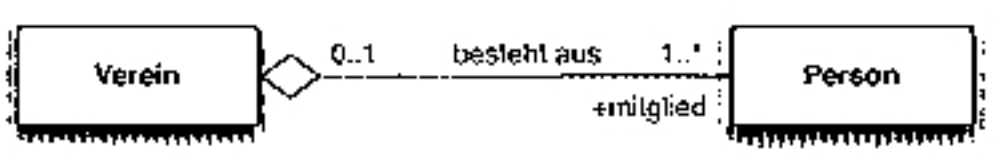
6.1.7 Aggregation

- Die Aggregation ist ein Spezialfall der allgemeinen Assoziation und beschreibt wie sich etwas Ganzes aus seinen Teilen logisch zusammensetzt.



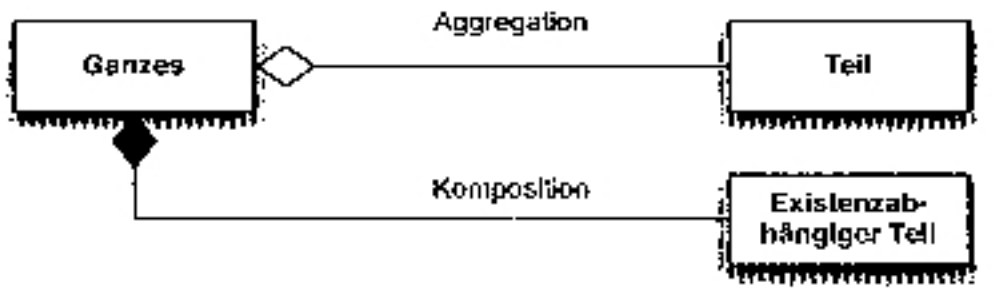
- Bei der Aggregation handelt es sich um eine asymmetrische Beziehung zwischen nicht gleichwertigen Partnern.

- Aggregationen werden durch eine Raute dargestellt, die jenes Ende der Assoziationskante markiert, das zur Aggregatklasse, also zum "Ganzen" hinführt.



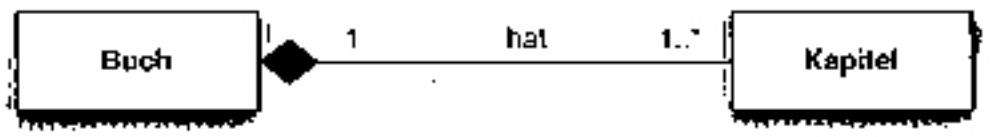
6.1.8 Komposition

- Die Komposition ist eine strengere Form der Aggregation, bei der die Teile vom Ganzen existenzabhängig sind. Sie beschreibt, wie sich etwas Ganzes aus Einzelteilen zusammensetzt und diese kapselt.



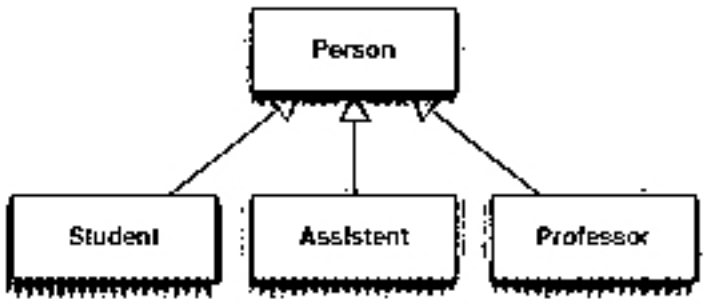
- Die Kardinalität auf der Seite des Aggregats muss immer 1 sein und jedes Teil darf nur Teil genau eines Kompositionsobjekts sein, sonst wäre die Existenzabhängigkeit widersprüchlich.

- Die Kompositioon wird durch eine ausgefüllte Aggregationsraute dargestellt.



6.1.9 Generalisierung

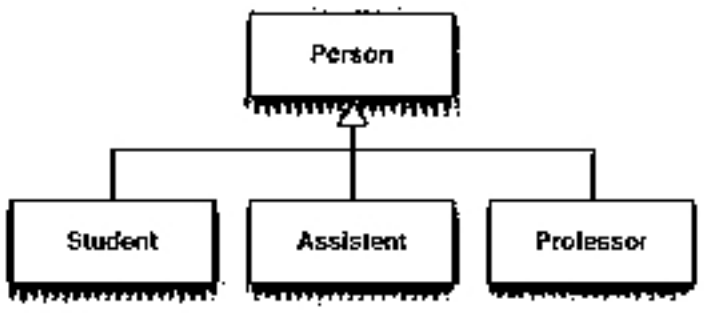
- Die Generalisierung stellt eine taxonomische Beziehung zwischen einer spezialisierten Klasse (Unter- oder Subklasse) und einer allgemeineren Klasse (Ober-, Basis- oder Superklasse) dar.



- Die Subklasse erbt die Charakteristika der Superklasse und kann weitere Merkmale hinzufügen.

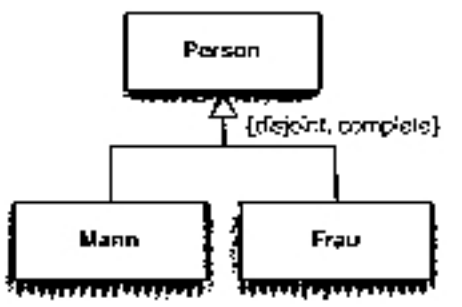
- Die Generalisierung stellt eine "is-a" Beziehung dar.

- Die Generalisierung wird durch einen Pfeil mit einem nicht ausgefüllten gleichseitigen Dreieck mit Spitze zur Superklasse notiert.
  Es können entweder mehrere Generalisierungspfeile eingezeichnet werden, oder diese können zu einem Mehrfachpfeil verschmelzen.

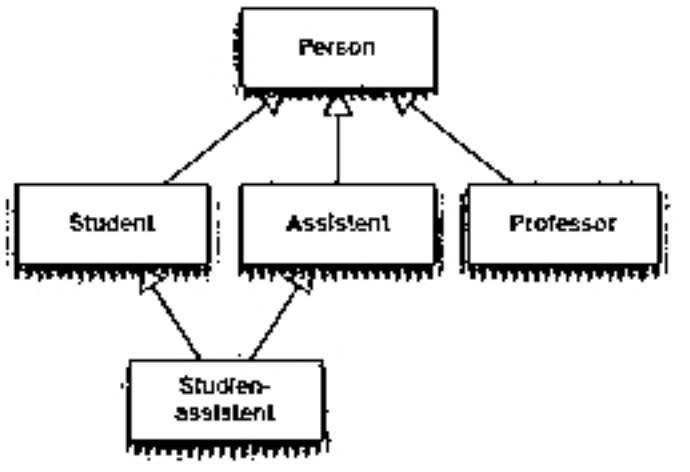


- Auf die Generalisierungsbeziehung können noch Einschränkungen getroffen werden, die neben das Dreieck in geschwungenen Klammern eingezeichnet werden.

- Somit kann über vollständig / unvollständig bzw. über überlappend / disjunkt eine Aussage getroffen werden.



- UML Klassendiagramme selbst unterstützen die Notation von Mehrfachvererbung.



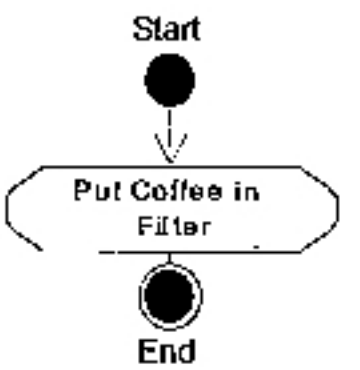
Literatur und Quellen:

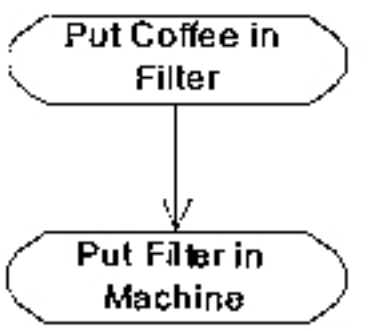
- Kapitel 3.1 in:
  Martin Hitz, Gerti Kappel, Elisabeth Kapsammer, Werner Retschitzegger:
  UML @ Work - Objektorientierte Modellierung mit UML2.
  dpunkt Verlag 2005 / 3. aktualis. u. überarb. Aufl. 2005.
  ISBN-13: 9783898642613 ISBN-10: 3898642615

- Kapitel 2.2 in:
  Bernd Österreich: UML Kurzreferenz für die Praxis

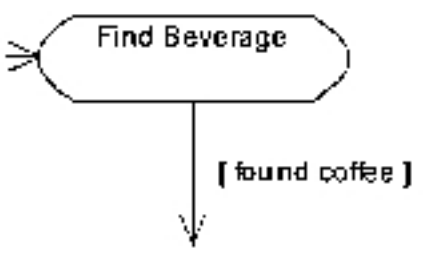
7 Aktivitätsdiagramm

- Aktivitätsdiagramme werden zur Modellierung eines Kontrollflusses genutzt.
- Die Modellierung von Prozessen erfolgt in einem Aktivitätsdiagramm durch die Angabe einer bestimmten Abfolge von Aktionen.
- Eine Aktion wird durch ein Rechteck mit konvexen Vertikalen dargestellt, das den Namen der auszuführenden Aktion enthält.
- Beginn und Ende des Ablaufs werden durch einen eindeutigen Startzustand und möglicherweise mehrere Endzustände markiert.
- Aktionen sind untereinander durch Kontrollflusskanten verbunden, welche die Ausführungsreihenfolge der einzelnen Schritte festlegen.

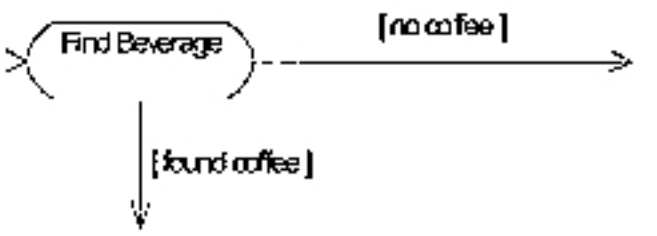




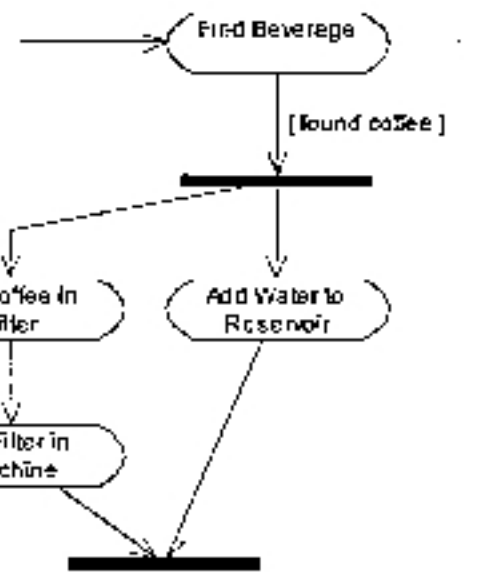
- Ein Kontrollfluss von einer Aktion A zur Aktion B wird durch einen (Übergangs-)Pfeil von A nach B dargestellt. Sobald die Ausführung von A abgeschlossen ist, wird automatisch der Übergang (Transition) nach B ausgelöst. Somit fungiert der Abschluss von A als impliziter Trigger für die Ausführung von B.
- Üblicherweise werden in Aktivitätsdiagrammen keine Ereignisse dargestellt, die als explizite Auslöser für Übergänge dienen.
- Jeder Aktion muss mindestens ein eingehender und ein ausgehender Übergang zugeordnet sein.
- Ein eingehender Übergang legt die Vorgängeraktivität fest. Ein ausgehender Übergang bestimmt die Nachfolgeraktivität.
- Eine Aktion kann beliebig viele eingehende und ausgehende Übergänge haben.
- Ein Übergang kann mit einer Überwachungsbedingung in eckigen Klammern versehen werden. In diesem Fall wird der Übergang nur dann ausgeführt, wenn die vorangegangene Aktion abgeschlossen und die Bedingung erfüllt ist.



- Sind einer Aktion mehr als ein ausgehender Übergang zugeordnet, werden diese in der Regel mit einander wechselseitig ausschließenden Bedingungen überwacht, um so indeterministische Zustandsübergänge zu vermeiden. Diese bedingten Übergänge ermöglichen die Modellierung alternativer Abläufe.



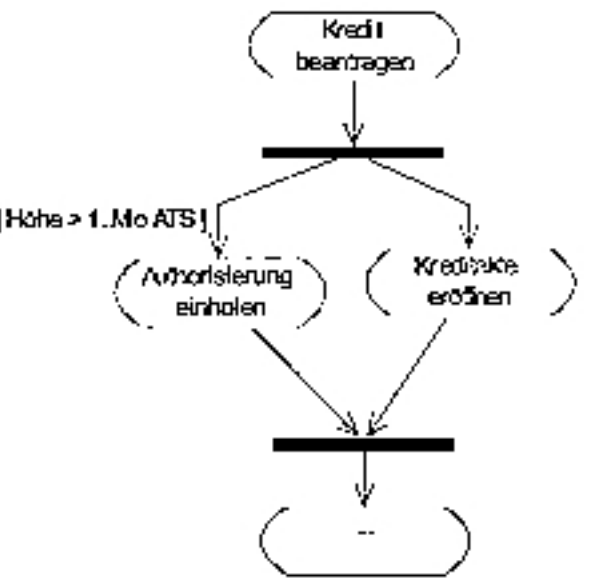
- Im Aktivitätsdiagrammen ist es auch möglich nebenläufige Teilfolgen von Aktionen zu beschreiben, die parallel abgearbeitet werden können. Dazu gibt es die Pseudozustände des Synchronisationsbalken und der Gabelung.



- Die Gabelung zeigt den Beginn einer Nebenläufigkeit durch einen Synchronisationsbalken an, von dem mehrere Übergänge wegführen.

- Die Vereinigung gibt das Ende einer Nebenläufigkeit an. Sie wird durch einen Synchronisationsbalken dargestellt, der zwei oder mehrere unbedingte eingehende Übergänge besitzt.

- Übergänge, die vom Synchronisationsbalken einer Gabelung ausgehen, können auch eine Überwachungsbedingung tragen. Sollte eine solche Bedingung zum Zeitpunkt des Schaltens des Übergangs nicht erfüllt sein, wird der zugehörige Zweig im Aktivitätsdiagramm nicht aktiviert.



- Sollte bei einer zugehörigen Gabelung auf Grund einer Überwachungsbedingung ein bestimmter Zweig nicht aktiviert worden sein, so ist dieser auch für die Vereinigung irrelevant und es wird nicht auf ihn gewartet.

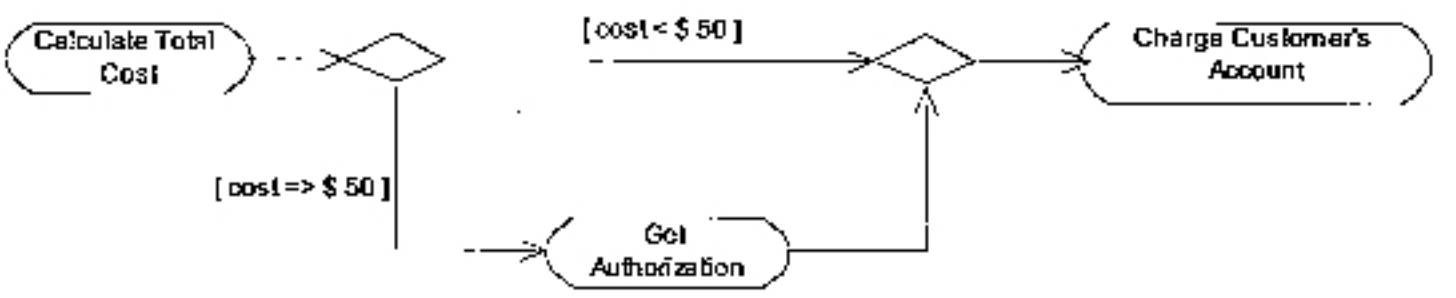
- Weiters können die Pseudozustände des Entscheidungsknoten und der Verbindungsstelle genutzt werden, um den sequentiellen Kontrollfluss besser zu strukturieren.

- Der Entscheidungsknoten ermöglicht es, alternative Abläufe explizit zu deklarieren und komplexe Entscheidungen in Form von Entscheidungsbäumen darzustellen.

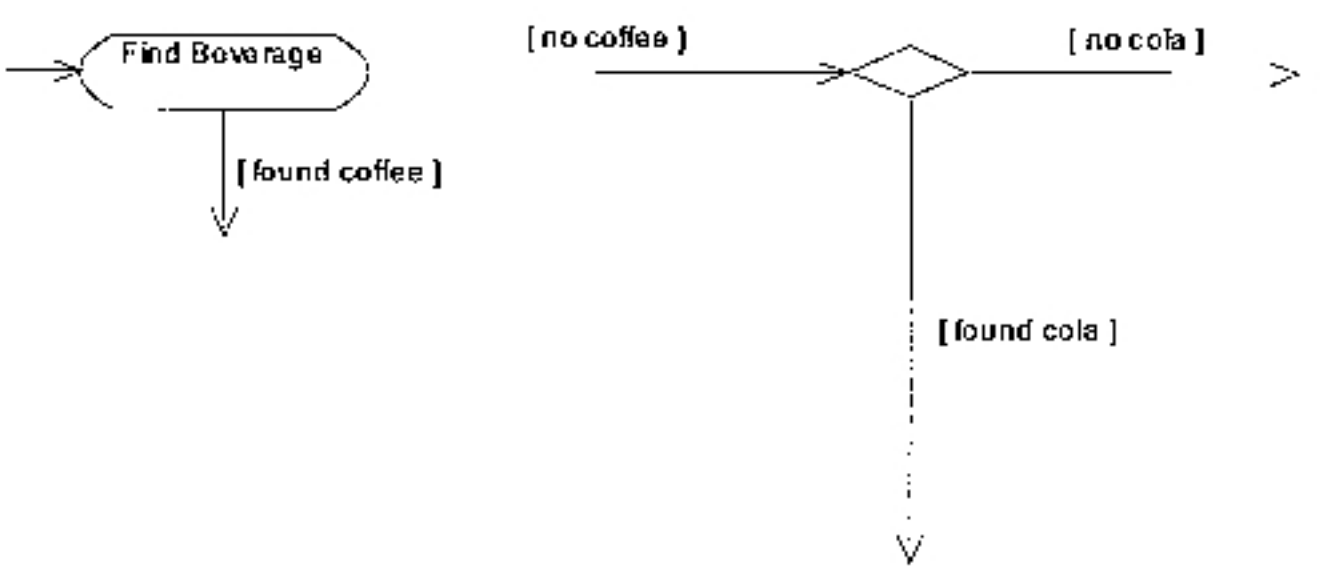
- Einem Entscheidungsknoten sind eine oder mehrere eingehende Übergänge und zwei oder mehrere ausgehende Übergänge zugeordnet. (Allfällige Ereignisse dürfen nur auf eingehende Übergänge angegeben werden).

- Alle ausgehenden Übergänge müssen einander wechselseitig ausschließende Bedingungen tragen. Eine ausgehende Bedingung kann dabei ein Wildcard [else] tragen.

- Ein Kontrollfluss, der durch Entscheidungsknoten aufgespalten wurde, kann auch durch dasselbe Symbol wieder vereinigt werden.In dieser Rolle wird der Entscheiungsknoten als *Merge* bezeichnet.



- Im Falle von einfachen - also keinen geschachtelten Entscheidungen - ist es eine Geschmacksfrage, ob unmittelbar ausgehende bedingte Übergänge oder ein dazwischengeschalteter Entscheidungsknoten mit ausgehenden bedingten Übergängenzur Spezifikation von alternativen Abläufen verwendet wird.

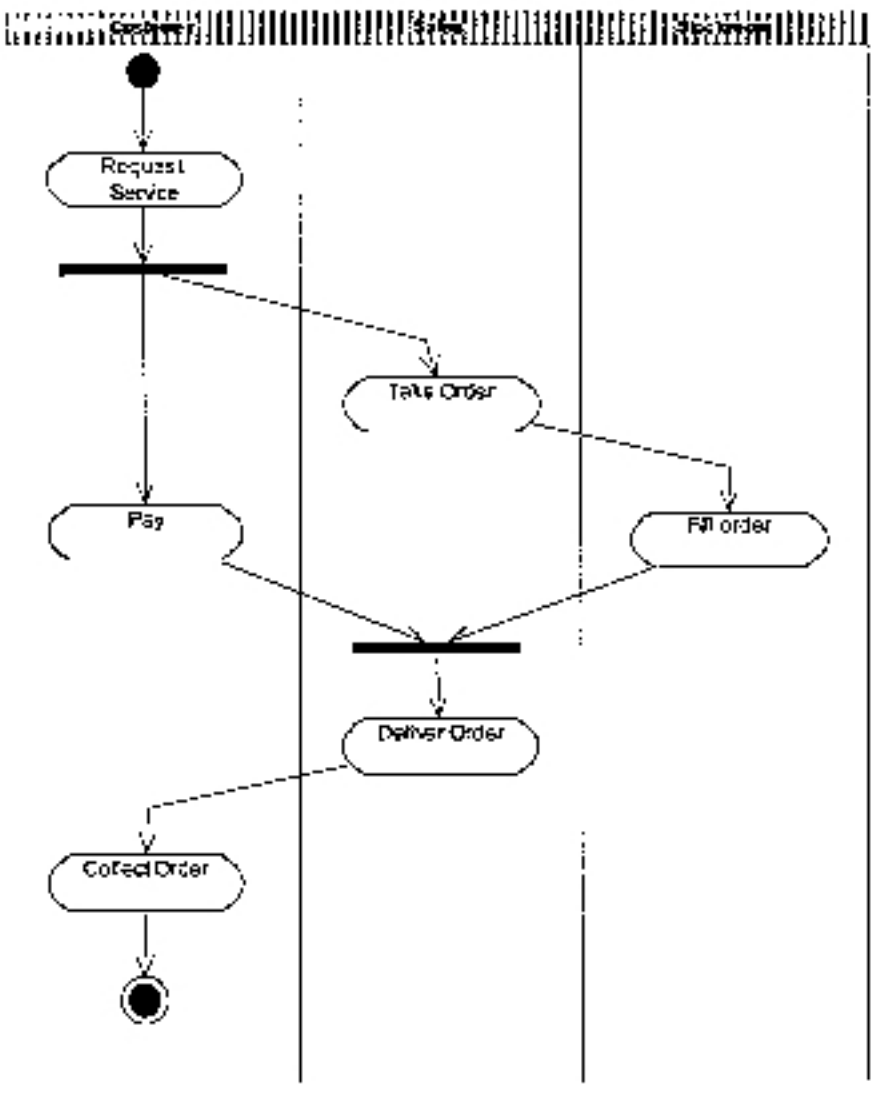


- Eine weitere Möglichkeit, den sequentiellen Kontrollfluss zu verzweigen und wieder zu vereinigen, stellen die Verbindungsstellen dar. Verbindungsstelle und Entscheidungsknoten sind im wesentlichen äquivalent. Daher werden Verbindungsstellen hier nicht weiter behandelt.

- Um die Aktionen in einem Aktivitätsdiagramm verantwortlichen Objekten zuzuordnen werden Verantwortlichkeitsbereiche (Swimlanes) genutzt.

- Einem Verantwortlichkeitsbereich sind einerseits bestimmte Aktionen und andererseits für die Ausführung der Aktionen verantwortliche Objekte zugeordnet. Diese Objekte sind entweder Objekte des zu realisierenden Systems, Akteure oder Organisationseinheiten eines Unternehmens, dessen Geschäftsprozesse beschrieben werden.

- Verantwortlichkeitsbereiche entsprechen Regionen im Aktivitätsdiagramm, die mit den jeweils verantwortlichen Objekten beschriftet sind und durch vertikale Linien voneinander getrennt werden. Ihre relative Lage zueinander ist bedeutungslos. Jede Aktion wird genau einem Verantwortlichkeitsbereich zugeordnet.



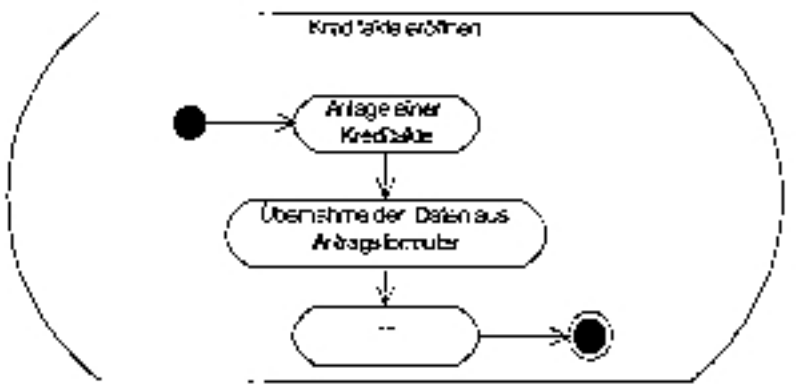
- Dem Aktivitätsdiagramm liegt als formales Modell der Aktivitätsgraph zugrunde. In diesem Aktivitätsgraph unterscheiden wir bei den Knoten zwischen den Aktionsknoten und den Aktivitätsknoten (zusätzlich zu den Pseudoknoten). Jedoch wird sowohl für Aktionen als auch Aktionen das gleiche Symbol verwendet.

- Eine Aktion ist atomar. D.h. sie kann nicht weiter zerlegt und während ihrer Ausführung nicht unterbrochen werden.

- Die Aktion weist eine komplexe Struktur auf, die ein eigener Aktivitätsgraph beschreibt.

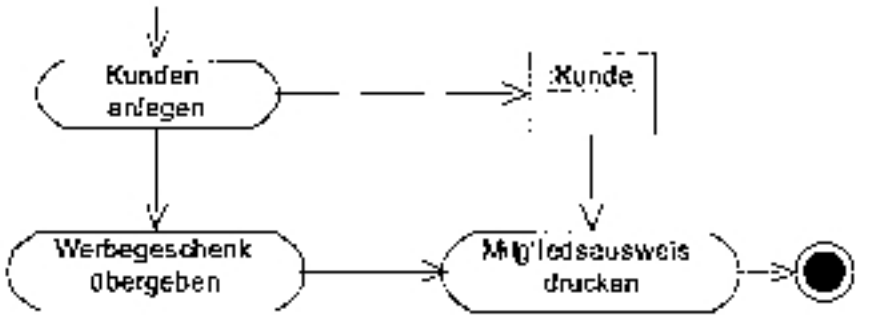
- Besitzt ein Knoten eine Verfeinerung, so handelt es sich konzeptionell um einen Aktivitätsknoten, ansonsten um einen Aktionsknoten.

- Die Verfeinerung kann graphisch in einem eigenen Diagramm, oder aber auch innerhalb des Symbols des Aktivitätsknoten angegeben werden.

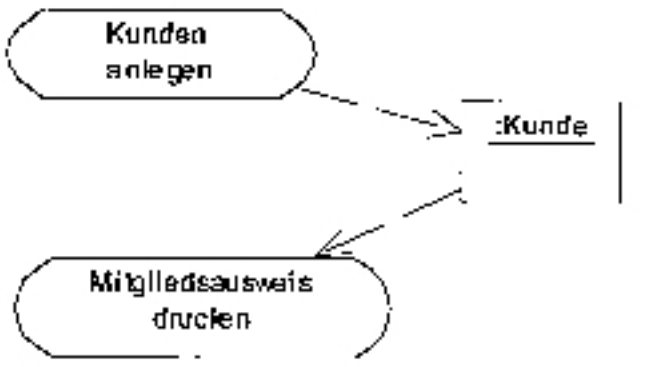


- Aktivitätsdiagramme können auch zur Modellierung von Objektflüssen genutzt werden.

- Objekte können als Eingabe für eine Aktion verwendet werden oder sie werden von einer Aktion manipuliert und als Ausgabe erzeugt.



- Objekte werden als Rechteck mit unterstrichenem Namen dargestellt. Sie werden mit den Aktionen durch eine gerichtete strichlierte Kante vom Objektsymbol zur verarbeitenden Aktion im Falle des Inputs bzw. im Falle des Outputs in umgekehrter Richtung dargestellt.



- Sollte ein Objektfluss den Kontrollfluss zwischen zwei Aktionen vollständig erklären, so kann auf die graphische Darstellung des Kontrollflusses verzichtet werden.

Literatur und Quellen:

- Kapitel 4.2 in:
  Martin Hitz, Gerti Kappel, Elisabeth Kapsammer, Werner Retschitzegger:
  UML @ Work - Objektorientierte Modellierung mit UML2.
  dpunkt Verlag 2005 / 3. aktualis. u. überarb. Aufl. 2005.
  ISBN-13: 9783898642613 ISBN-10: 3898642615

8 Zustandsdiagramm

Inhalt

8.1 Allgemeines ..... -1
8.2 Darstellung von Zustandsdiagrammen ..... -3
8.3 Ereignisse ..... -10
8.4 Verfeinerung ..... -11
8.5 Komplexer Übergang ..... -16
8.6 Segmentierter Übergang ..... -17
8.7 Gesamtbeispiel Fernseher ..... -18
8.8 Exkurs: Zustandsdiagramme vs. Aktivitätsdiagramme ..... -19

8.1 Allgemeines

- Zustandsdiagramme basieren auf den Konzepten von endlichen Automaten.
- Endliche Automaten entsprechen Maschinen deren Zustandsverhalten nicht nur eine Folgerung des aktuellen Inputs, sondern auch die Folge mehrerer Inputs aus der Vergangenheit ist.
- Jeder endliche Automat ist durch einen derzeitigen, aktuellen Zustand gekennzeichnet, der das Ergebnis der Vergangenheit festhält.
- Zustandsdiagramme beschreiben die möglichen Folgen von Zuständen, die ein Objekt
  - während seines "Lebenslaufs", also von seiner Konstruktion bis zu seiner Destruktion, sowie
  - während der Ausführung einer Operation

  einnehmen kann.

- Ein Zustandsdiagramm stellt einen Graph dar, dessen Knoten den Zuständen entsprechen, die von Objekten eingenommen werden können; z.B.: eine Lampe kann die Zustände "Licht aus" und "Licht an" einnehmen.

- Die gerichteten Kanten stellen die möglichen Zustandsübergänge dar; z.B. der Übergang von "Licht aus" zu "Licht an".

- Die Zustandsübergänge werden normalerweise von exogenen Stimuli ausgelöst, die als Ereignisse bezeichnet werden; z.B. Betätigung des Lichtschalters.

8.2 Darstellung von Zustandsdiagrammen

- Ein Zustand wird im Diagramm durch ein Rechteck mit abgerundeten Ecken dargestellt. Dieses Rechteck kann eventuell in zwei Abschnitte unterteilt sein.

- Im Normalfall wird ein Zustand durch seinen Namen im oberen Abschnitt charakterisiert. Die Angabe eines Namens könnte aber auch entfallen, denn auch zwei unbenannte Zustände gelten grundsätzlich als voneinander verschieden.

- Der Begriff "Zustand eines Objektes" wird oft mit den Ausprägungen der Attribute und Assoziationen eines Objektes zu einem bestimmten Zeitpunkt gleichgesetzt. Im Kontext von Zustandsdiagrammen verwenden wir jedoch einen weitläufigeren Zustandsbegriff.

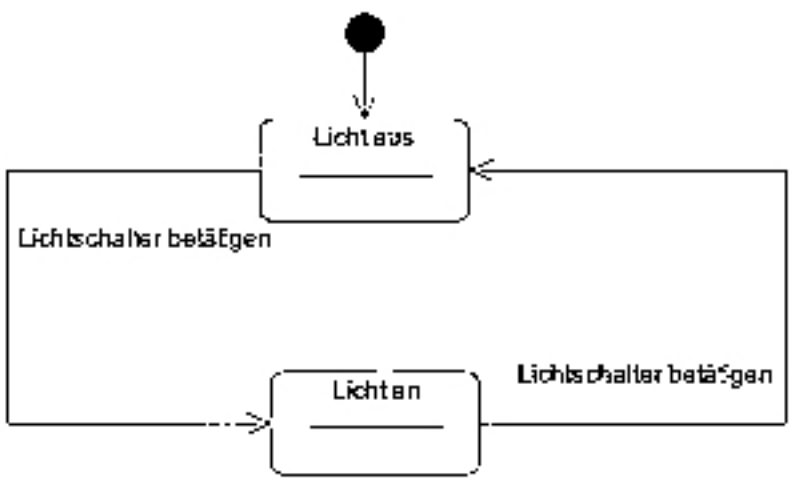
- Ein Zustand bezeichnet eine Situation in der ein betrachtetes Objekt ein "qualitativ einheitliches Verhalten" aufweist. Ein Objekt kann in einem gewissen Zustand auf äußere Ereignisse auf eine situationsspezifische (= zustandsspezifische) Weise reagieren, wobei diese Reaktion von Fall zu Fall (von Zustand zu Zustand) unterschiedlich sein kann.

- Objekte verweilen eine gewisse Zeit lang in einem bestimmten Zustand, bis ein Ereignis einen Zustandsübergang auslöst, der dann augenblicklich erfolgt ohne selbst Zeit in Anspruch zu nehmen.

- Zuständen können daher andauernde Aktivitäten zugeordnet werden, die ausgeführt werden, solange sich das Objekt im jeweiligen Zustand befindet.

- Übergänge (Transitionen) hingegen können mit sogenannten Aktionen assoziiert werden. Aktionen sind atomare Tätigkeiten, die im Zuge des Zustandsübergangs durchgeführt werden und (konzeptionell) ebenfalls keine Dauer aufweisen.

- Übergänge (Transitionen) werden durch einen i.a. beschrifteten Pfeil dargestellt, der einen Zustand (Quell- oder Vorzustand) mit dessen Folgezustand (Ziel- oder Nachzustand) verbindet.

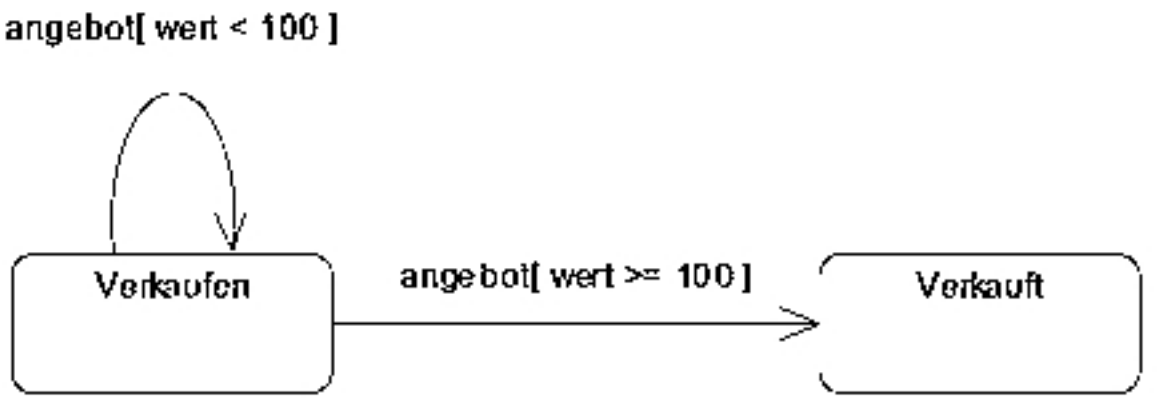


- Die Beschriftung eines Überganges kann folgende Angaben enthalten:
  - auslösendes Ereignis
  - optionale Bedingung
  - Aktionen

- Angabe des auslösenden Ereignisses
  - kann durch Argumente näher beschrieben sein.
  - Sobald das Ereignis eintritt (und keine zusätzliche Bedingung angegeben ist), erfolgt der Übergang in den Folgezustand.

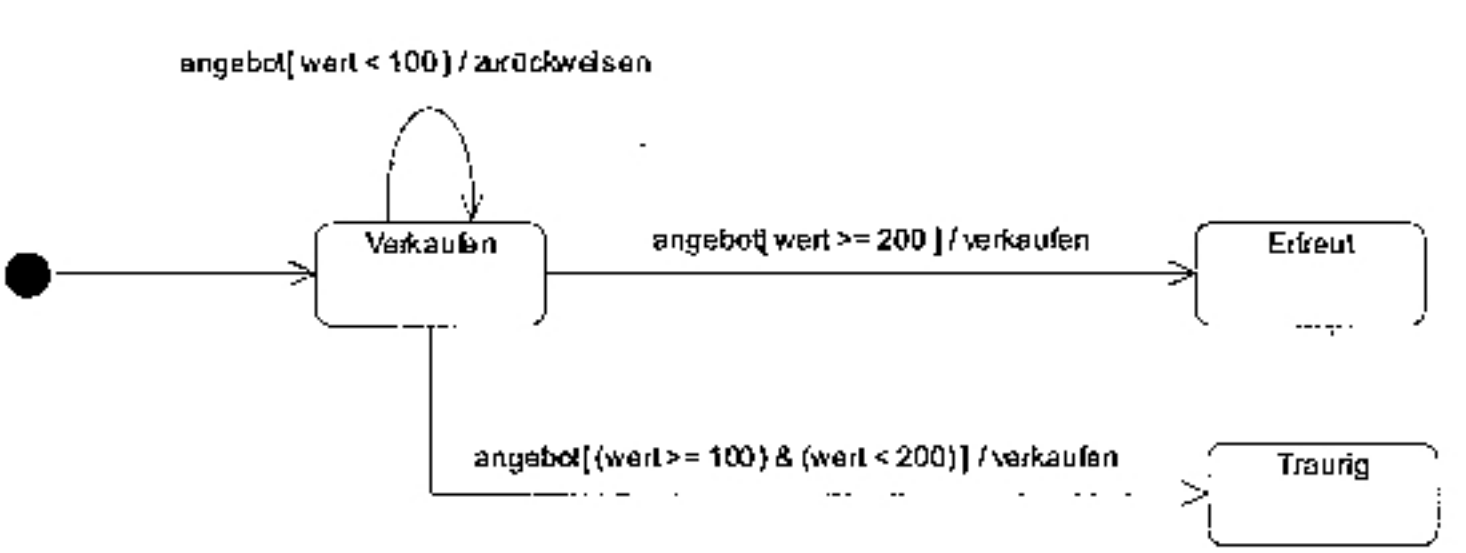
- Angabe einer optionalen Bedingung zur Überwachung des Übergangs.
  - Sobald das Ereignis eintritt und die Bedingung erfüllt ist, erfolgt der Übergang in den Folgezustand.
  - Tritt das Ereignis ein und ist die Bedingung nicht erfüllt, so erfolgt kein Zustandsübergang. Das "nicht konsumierte" Ereignis geht verloren: Selbst wenn die Bedingung später einmal eintritt, muss das Ereignis noch einmal auftreten um zum Übergang zu führen.

- Mit Hilfe einander wechselseitig ausschließender Bedingungen ist es möglich, dass ein und dasselbe Ereignis mehreren Transitionen eines Zustands zugeordnet ist. Tritt das Ereignis ein, so schaltet jener Übergang dessen Überwachungsbedingung erfüllt ist.



- Zur Angabe von komplexen Bedingungen können wie bei Aktivitätsdiagrammen Entscheidungsknoten genutzt werden.

- Ein Übergang kann Aktionen spezifizieren, die im Zuge des Schaltens des Überganges ausgeführt werden. Diese Aktionen können unterschiedlicher Natur sein und in Programmiersprache bzw. in Pseudocode formuliert sein.

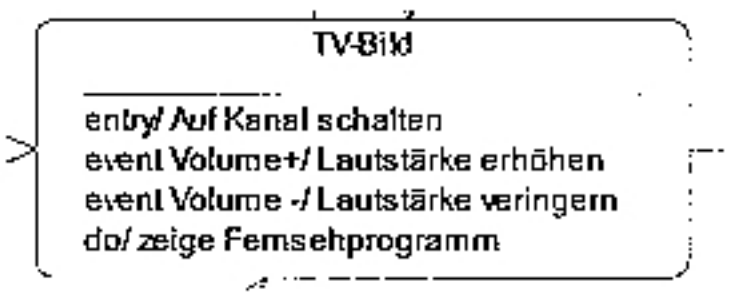


- Innerhalb eines Zustandes können (im unteren Abschnitt des graphischen Symbols) Angaben über Aktivitäten und sogenannte innere Übergänge enthalten.

- Eine Aktivität innerhalb eines Zustandes wird nach dem Präfix do/ angegeben (eventuell mit einer Parameterliste) und kann eventuell durch ein anderes Zustandsdiagramm näher beschrieben werden.

- Innere Übergänge werden - wie normale Übergänge - von Ereignissen ausgelöst. Sie verlassen aber den aktuellen Zustand nicht. Ihre Spezifikation erfolgt daher innerhalb des Zustandsymbols.

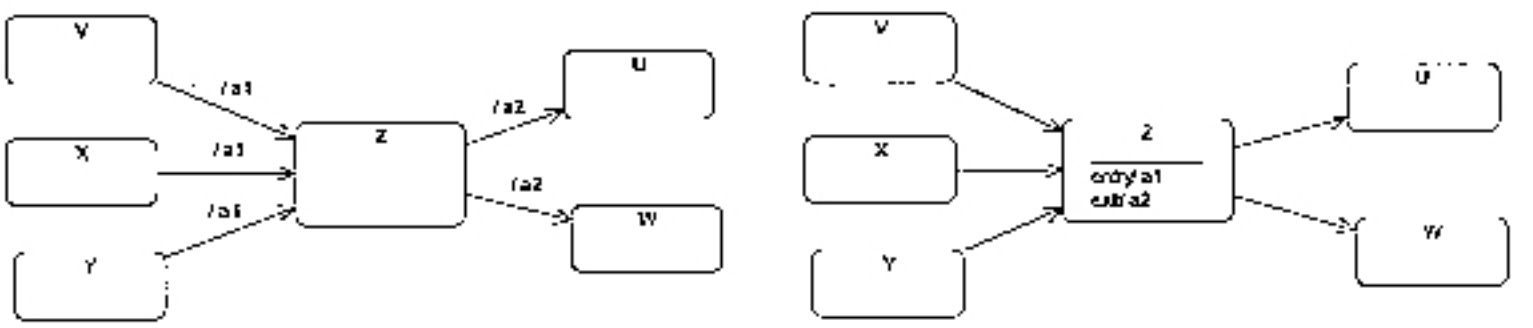
- Bei inneren Übergängen wird zuerst das Ereignis und dann getrennt durch einen Schrägstrich "/" die Aktion spezifiziert. Dabei können dem Ereignis innerhalb von Klammern eine Argumentenliste und innerhalb von eckigen Klammern bestimmte Überwachungsbedingungen folgen.



- Bei inneren Übergängen können auch die beiden Pseudoereignisse "entry" und "exit" angegeben werden:
  - Nach "entry /" werden Aktionen angegeben, die immer ausgeführt werden, wenn das Objekt in den Zustand eintritt.
  - Nach "exit /" werden Aktionen angegeben, die immer ausgeführt werden sobald das Objekt den Zustand verlässt.

  Diese beiden Pseudoereignisse werden dann genutzt, wenn mehr als ein Übergang in den Zustand führt bzw. den Zustand verlässt und bestimmte Aktionen in allen Fällen durchzuführen sind.

- Sollte ein und dasselbe Ereignis sowohl eine innere als auch eine äußeren Übergang markieren, so wird nur der innere Übergang anstelle des äußeren Übergangs durchgeführt.



- In Zustandsdiagrammen können folgende Pseudozustände genutzt werden: Startzustand, Entscheidungsknoten, Synchronisationsbalken, sowie Verbindungsstellen, History-Zustände und Synch-Zustände.

- Als Gegenstück zum Startzustand gibt es in Zustandsdiagrammen auch einen Endzustand (bzw. eventuell mehrere). Der Endzustand ist jedoch kein Pseudozustand, da es dem Objekt möglich sein muss in ihm zu verweilen.

8.3 Ereignisse

- Ereignisse können unterschiedlicher Natur sein. Ein Ereignis ist in UML als "bemerkenswerte Begebenheit" definiert. Es werden folgende Varianten an Ereignissen unterschieden:
  - SignalEvent
  - CallEvent
  - ChangeEvent
  - TimeEvent

- Bei einem SignalEvent wird ein Signal von einem anderen Objekt empfangen. Der Ereignisname bezeichnet das Signal.

- Beim CallEvent wird eine Nachricht im Sinne eines Operationsaufrufs empfangen. Der Ereignisname entspricht dem Namen der aufgerufenen Operation.

- Beim ChangeEvent wird eine bestimmte Bedingung erfüllt. Statt eines Ereignisnamens wird das Schlüsselwort "when" und die Bedingung als logischer Ausdruck angeführt. Das Ereignis tritt ein, sobald der Wert des logische Ausdrucks von falsch auf wahr wechselt.

- Ein TimeEvent tritt ein falls eine vorgegebene Zeitspanne seit einem bestimmten Ereignis abgelaufen ist. Dabei wird nach dem Schlüsselwort "after" die konstante Zeitspanne und ein optionales Bezugsereignis angegeben.

8.4 Verfeinerung

- Aktivitäten eines Zustandsdiagramms können durch ein weiteres Zustandsdiagramm beschrieben werden. Dies entspricht der Aktivierung eines geschachtelten Automaten.

- Die Zuordnung eines Zustandsdiagramms zu einer Aktivität eines Zustands stellt eine Verfeinerung von Zustandsdiagrammen dar. In UML wird unterschieden zwischen:
  - ODER-Verfeinerung
  - UND-Verfeinerung

- Bei der ODER-Verfeinerung, in der ein komplexer Zustand in Subzustände zerlegt wird, befindet sich ein Objekt immer in genau einem der Subzustände, sobald es sich im komplexen Zustand befindet.

- Die ODER-Verfeinerung kommt in zwei Varianten vor:
  - Angabe eines eigenen, getrennten Zustandsdiagramms, das den komplexen Zustand verfeinert. Diese Variante wird meist benutzt um einem komplexen Zustand zugeordnete Aktivitäten (do / ...) näher zu beschreiben.

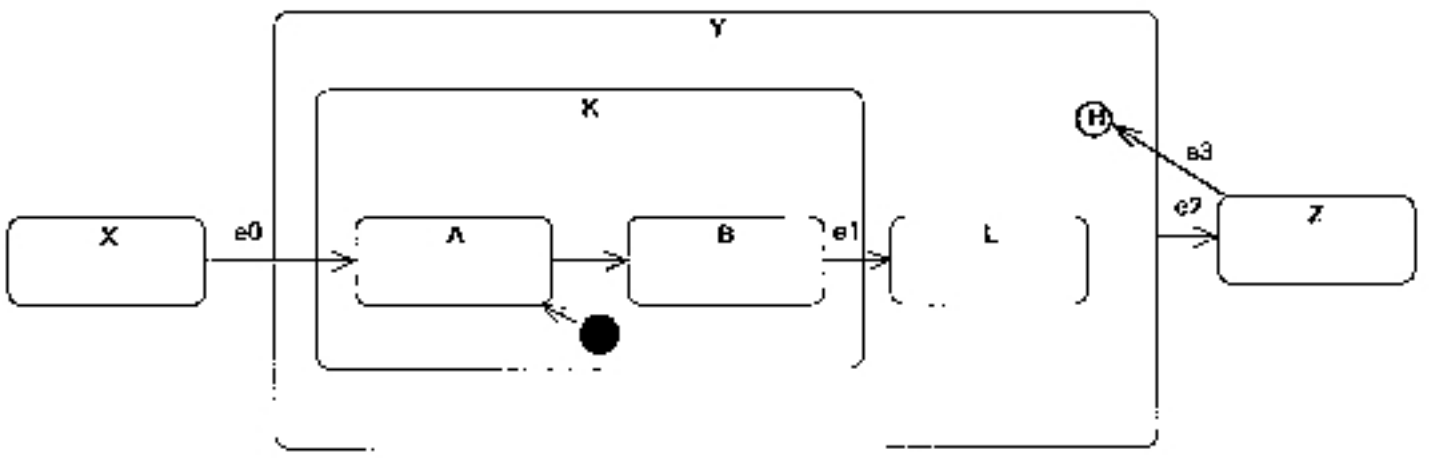
  - Das erweiternde Zustandsdiagramm kann innerhalb des Zustandsymbols des komplexen Zustands angegeben werden. Dazu kann das Zustandssymbol des komplexen Zustands einen weiteren Abschnitt beinhalten.

- Bei einer ODER-Verfeinerung kann der Pseudozustand "History-Zustand" genutzt werden, um bei Wiedereintritt in einen komplexen Zustand in den zuletzt aktiven Subzustand wieder einzutreten.

- Der mit einem History-Zustand ausgestattete ODER-verfeinerte Zustand führt Buch über den zuletzt aktiven Subzustand, der bei einem Übergang in den History-Zustand wieder angenommen wird.

- Sollte von einem History-Zustand ein Übergang ausgehen, so wird dieser nur verfolgt, wenn es sich um den erstmaligen Eintritt in den komplexen Zustand handelt.

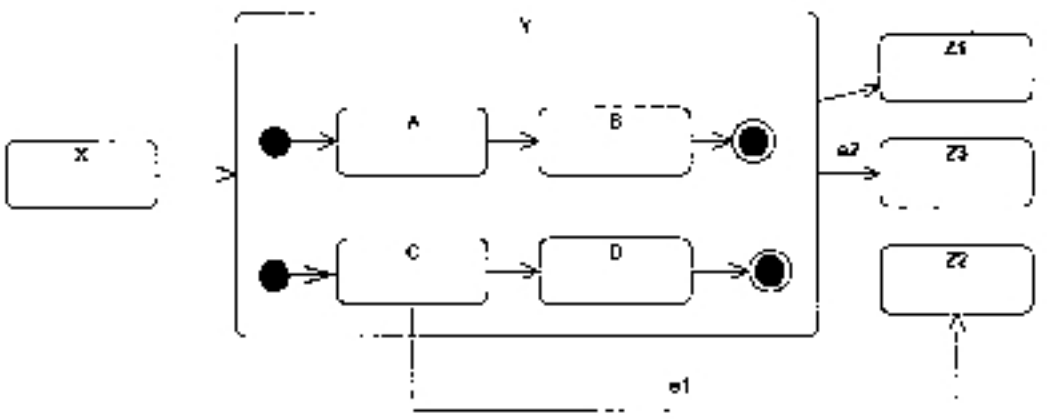
- Ein History-Zustand bezieht sich normalerweise auf alle Zustände der obersten Verfeinerungsstufe innerhalb des komplexen Zustands, der den History-Zustand beherbergt. Wenn in einen Subzustand beliebiger Schachtelungstiefe zurückgegangen werden soll, muss ein "tiefer" History-Zustand verwendet werden.



- Bei UND-Verfeinerung wird ein komplexer Zustand in nebenläufige, gleichzeitig aktive Subzustände verfeinert. Jede dieser Subzustandsfolgen wird durch eine strichlierte Linie voneinander getrennt im Zustandssymbol des komplexen Zustands angegeben. Jede dieser Regionen kann benannt sein und enthält ein eigenes Zustandsdiagramm.

- Ein Übergang in den komplexen Zustand einer UND-Verfeinerung bedingt den Start der jeweiligen Subzustandsfolgen.

- Ein UND-verfeinerter Zustand wird auf zwei alternative Arten verlassen:
  - Sobald die Endzustände aller Subzustandsfolgen erreicht worden sind, wird der komplexe Zustand über eine eindeutigen und unmarkierten Übergang verlassen.
  - Existiert ein Übergang von einem Subzustand der aus den komplexen Zustand hinaus führt, so wird der komplexe Zustand verlassen, sobald der entsprechende Übergang schaltet. Dies erfolgt ohne Rücksicht auf die anderen nebenläufigen Subzustände, die dadurch ebenfalls verlassen werden.



8.5 Komplexer Übergang

- Bei der UND-Verfeinerung werden alle Subzustandsfolgen gleichzeitig aktiviert und starten mit ihrem "Startzustand".

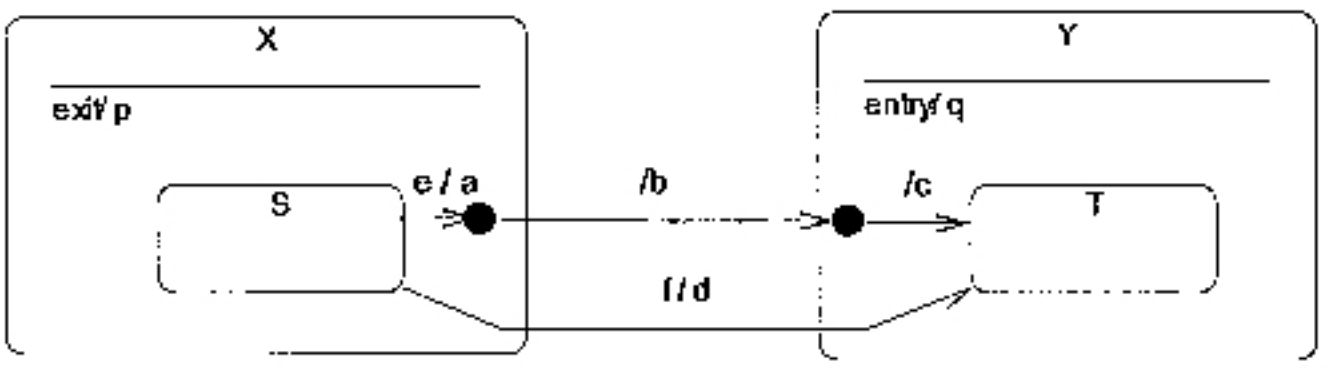
- Eine Zustandsfolge kann jedoch auch mit Hilfe von Synchronisationsbalken (siehe Aktivitätsdiagramme) aufgespalten werden.

- Diese Form von komplexen Übergängen ist jedoch bei Zustandsdiagrammen weniger üblich als bei Aktivitätsdiagrammen.

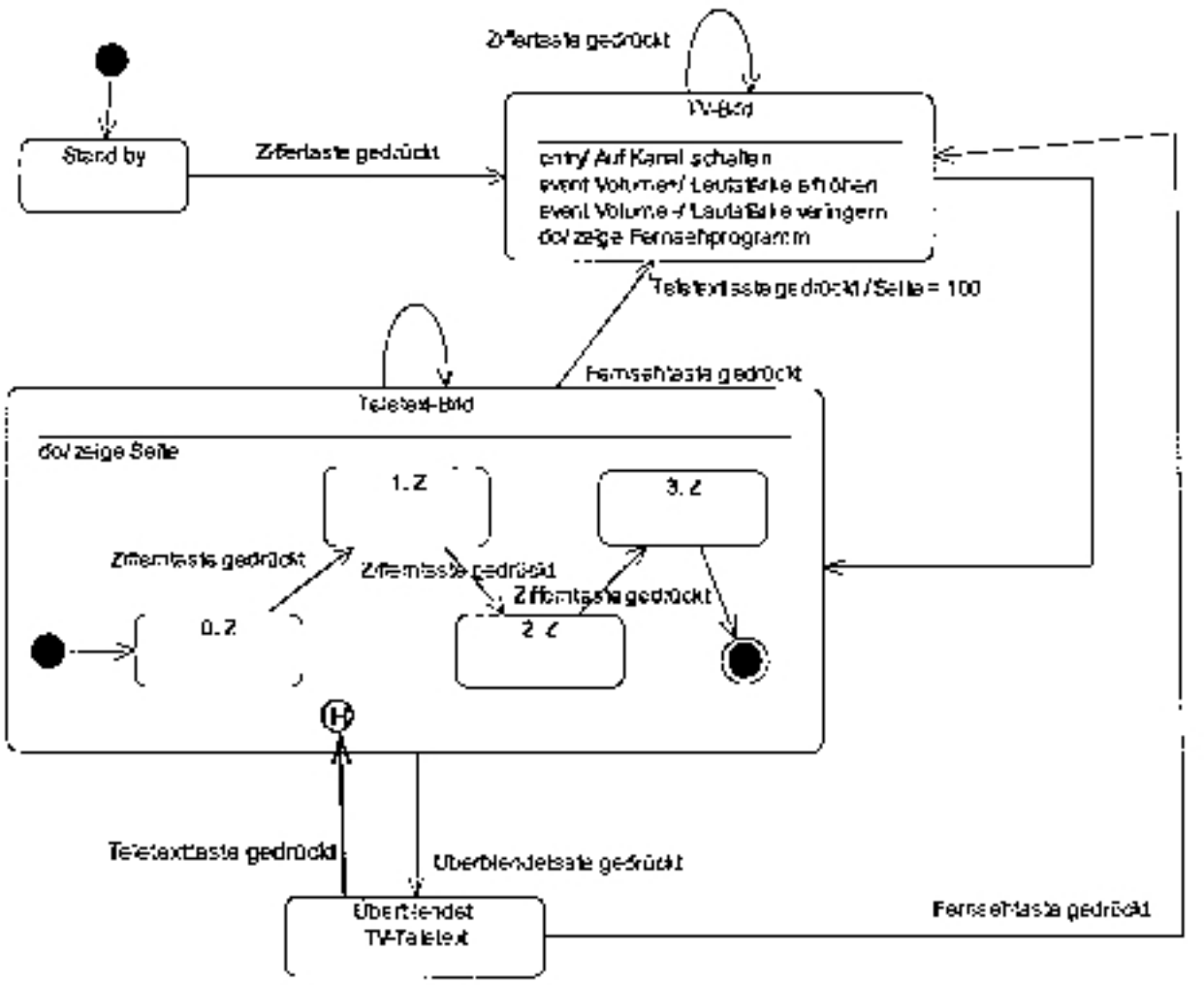
8.6 Segmentierter Übergang

- Auch bei sequentiellen (nicht-verzweigenden) Zustandsfolgen können mehrere einzelne Übergänge zu einem komplexen Übergang zusammengefasst werden.

- Die einzelnen Segmente (oder Teilübergänge) werden durch Verbindungsstellen miteinander verknüpft und können jeweils Überwachungsbedingungen und Aktionen tragen. Von einer Verbindungsstelle ausgehende Übergänge dürfen jedoch keine Ereignisse markieren.



8.7 Gesamtbeispiel Fernseher



8.8 Exkurs: Zustandsdiagramme vs. Aktivitätsdiagramme

- Modellierung von Prozessen kann erfolgen an Hand von:
  - Kontrollfluss
  - Datenfluss bzw. Objektfluss
  - Automaten

- Der Unterschied in den drei Modellen ist die Art und Weise wie ein Prozess initiiert wird

- Kontrollfluss:
  - Jeder Schritt (Aktivität) wird erst nach Vollendung des vorhergehenden Schrittes gestartet
  - ... unabhängig von den vorhandenen Inputs.
  - Der Schwerpunkt liegt auf der Reihenfolge der einzelnen Schritte.

- Datenfluss bzw. Objektfluss:
  - Jeder Schritt wird durchgeführt, wenn alle Inputdaten vorhanden sind
  - ... und nur wenn alle Inputdaten vorhanden sind.
  - Schwerpunkt liegt auf den Daten die zwischen den Schritten fließen

- Endliche Automaten:
  - Jeder Schritt wird durchgeführt, wenn vom Automaten ein Hievend erkannt wird
  - ... unter Verwendung der Inputs, die der Hievend liefert
  - Schwerpunkt liegt auf der Reaktion auf Ereignisse

Literatur und Quellen:

- Kapitel 4.3 in:
  Martin Hitz, Gerti Kappel, Elisabeth Kapsammer, Werner Retschitzegger:
  UML @ Work - Objektorientierte Modellierung mit UML2.
  dpunkt Verlag 2005 / 3. aktualis. u. überarb. Aufl. 2005.
  ISBN-13: 9783898642613 ISBN-10: 3898642615

9 Sequenzdiagramm

Inhalt

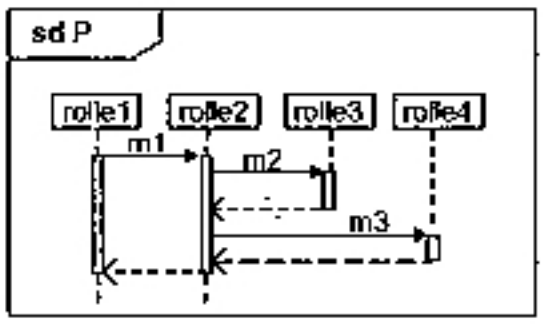
9.1 Exkurs Interaktionsdiagramme ..... -2
9.2 Übersicht Sequenzdiagramm ..... -6
9.3 Digrammrahmen ..... -8
9.4 Lebenslinie ..... -9
9.4.1 Ereignisspezifikation ..... -10
9.4.2 Reihenfolge ..... -11
9.4.3 Ausführungsspezifikation ..... -12
9.4.4 Aktives Objekt ..... -13
9.5 Nachricht ..... -14
9.5.1 Spezielle Nachrichtenarten ..... -15
9.5.2 Beispiel ..... -16
9.5.3 Zeiteinschränkungen ..... -17
9.5.4 Beispiel ..... -18
9.6 Zustandsinvariante ..... -19
9.7 Kombinierte Fragmente ..... -20
9.7.1 Notation ..... -21
9.7.2 Operatorarten ..... -22
9.7.3 Verzweigungen und Schleifen: alt-Operator ..... -23
9.7.4 Verzweigungen und Schleifen: loop-Operator ..... -24
9.7.5 Verzweigungen und Schleifen: opt- und break-Operator ..... -25
9.7.6 Parallelität und Ordnung: seq-Operator / strict-Operator ..... -26
9.7.7 Parallelität und Ordnung: par-Operator ..... -27
9.7.8 Parallelität und Ordnung: critical-Operator ..... -28
9.7.9 Filterungen und Zusicherungen: ignore-Operator / consider-Operator ..... -29
9.7.10 Filterungen und Zusicherungen: assert-Operator / neg-Operator ..... -30
9.8 Modularisierung ..... -31
9.8.1 Interaktionsreferenz ..... -31

9.8.2 Fortsetzungsmarke ..... -33
9.8.3 Verknüpfungspunkt ..... -34
9.9 Exkurs: Sequenzdiagramm vs. Aktivitätsdiagramm ..... -36

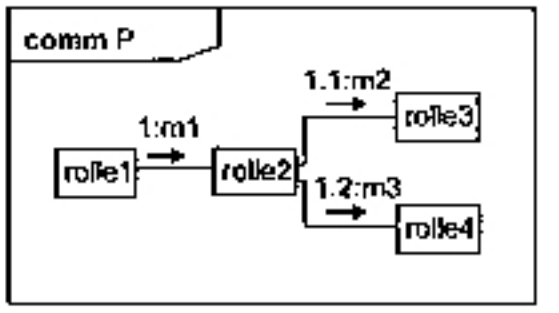
9.1 Exkurs Interaktionsdiagramme

- 4 Arten von Interaktionsdiagrammen
  - Für einfache Interaktionen **semantisch äquivalent**
  - Betonung **unterschiedlicher Aspekte**

- **Sequenzdiagramm** zeigt den zeitlichen und logischen Nachrichtenfluss
  - Reihenfolge von Nachrichten grafisch ersichtlich
  - Zeit ist **eigene** Dimension

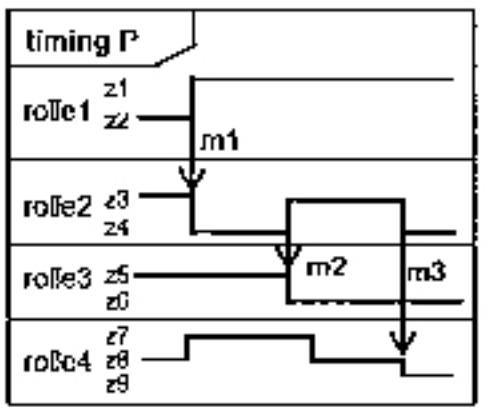


- **Kommunikationsdiagramm** ist »strukturell« orientiert
  - Zeigt die Beziehungen zwischen Interaktionspartnern – Kontextaspekt
  - Reihenfolge von Nachrichten nur über Dezimalklassifikation ausgedrückt
  - Zeit ist **keine eigene** Dimension

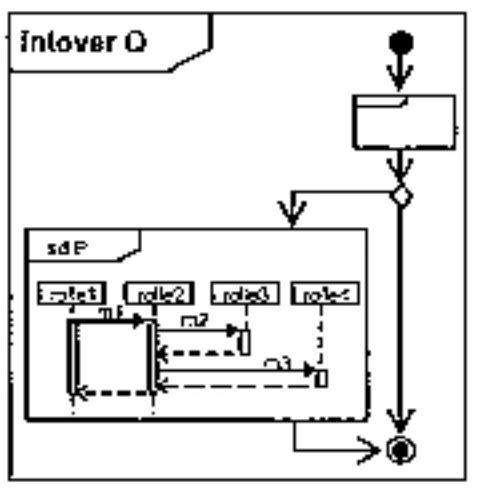


Exkurs Interaktionsdiagramme

- **Zeitdiagramm** zeigt Zustandsänderungender Interaktionspartner aufgrund von Zeitereignissen
  - Vertikale Dimension repräsentiert Interaktionspartner und ihre möglichen Zustände
  - Horizontale Dimension repräsentiert die Zeitachse



- **Interaktionsübersichtsdiagramm** zeigt das Zusammenspiel von verschiedenen Interaktionen
  - Visualisiert in welcher Reihenfolgeund unter welchen Bedingungen Interaktionsabläufe stattfinden



Exkurs Interaktionsdiagramme

- Zeigen wie **Nachrichten** zwischen verschiedenen **Interaktionspartnern** in einem bestimmten Kontext ausgetauscht werden

- Einsatzbereiche
  - Modellierung der **Interaktionen eines Systems mit seiner Umwelt** (Systemgrenzen festlegen, System als Black-Box)
  - Modellierung der **Realisierung eines Anwendungsfalls**
  - Modellierung des **Zusammenspiels der internen Struktur** einer Klasse, Komponente oder Kollaboration
  - Modellierung der Spezifikation von **Schnittstellen zwischen Systemteilen** (Zusammenspiel angebotene/benutzte Schnittstelle)
  - Modellierung der **Operationen einer Klasse**

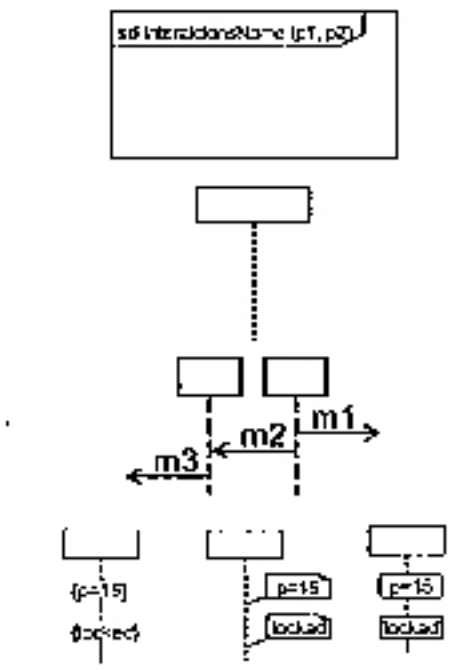
Exkurs Interaktionsdiagramme: Typ- vs. Instanzebene

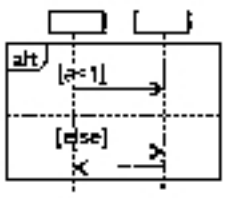
- **Modellierung** des Nachrichtenaustauschs zwischen Rollen und damit prinzipiell auf **Rollenebene**
  - Kontext der Interaktion durch **strukturierte Classifier** festgelegt = Kontext-Classifier
  - Deren **Rollen** stellen die **Interaktionspartner** dar
  - Tatsächliche Interaktion findet selbstverständlich auf **Instanzebene** zwischen **Objekten** statt

- **Modellierung** auf **Instanzebene möglich**, um eine Abfolge von Nachrichten zwischen konkreten Objekten darzustellen = **Trace**

9.2 Übersicht Sequenzdiagramm

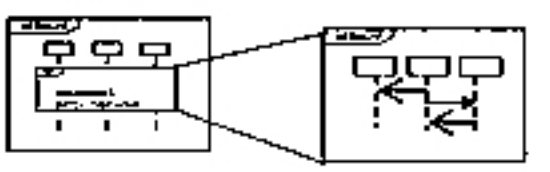
- Diagrammrahmen
- Lebenslinie
- Nachricht
- Zustandsinvariante
- Kombiniertes Fragment

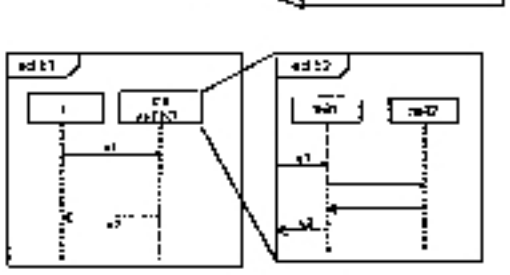


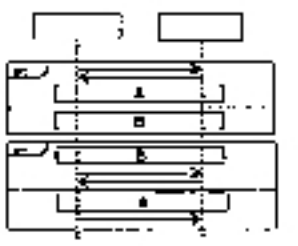


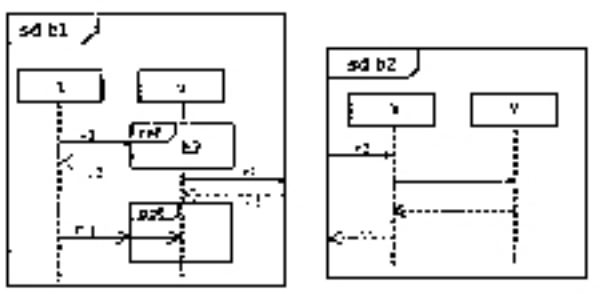
Darstellung von Sequenzdiagrammen

- Interaktionsreferenz
- Zerlegung einer Lebenslinie
- Fortsetzungsmarke
- Verknüpfungspunkt



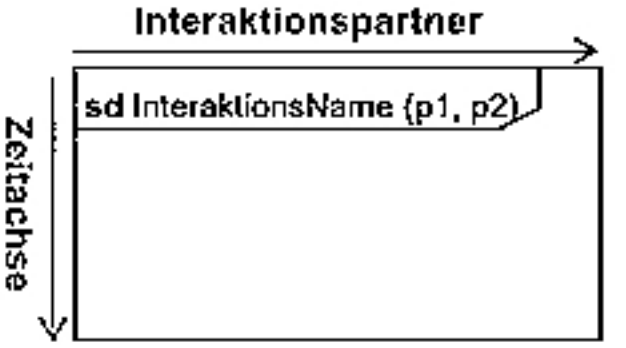






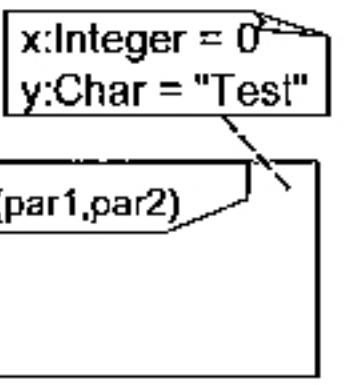
9.3 Digrammrahmen

- Rahmennotation
  - Für alle UML2 Diagrammarten möglich
  - Pentagon
    - Diagrammtypsd für Sequenzdiagramm
    - Interaktionsname
    - Optionale Parameter



- Zwei Darstellungsdimensionen
  - Vertikale Dimension repräsentiert Zeitachse
  - Horizontale Dimension repräsentiert Interaktionspartner in Form von Rollen

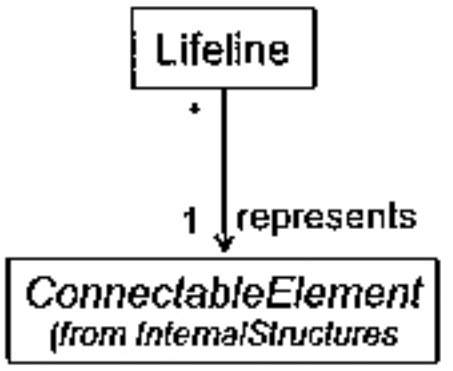
- Lokale Attribute
  - zwei Varianten:



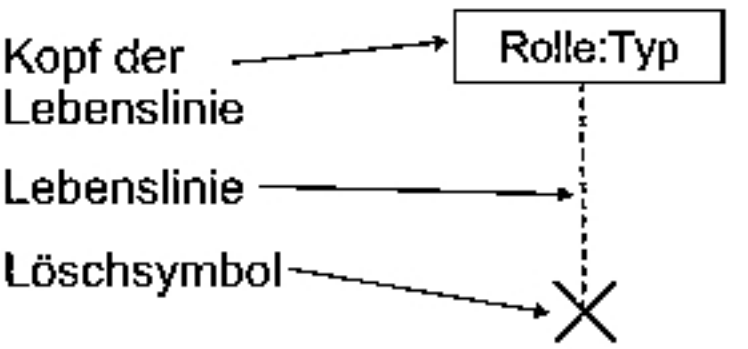
9.4 Lebenslinie

- Eine Lebenslinie beschreibt **genau einen Interaktionspartner**
- Als Interaktionspartner können **alle Rollen** des **Kontext-Classifiers** auftreten
- Rollen sind vom Typ **ConnectableElement** (z.B. Klassen, Attribute oder Ports)

Metamodell



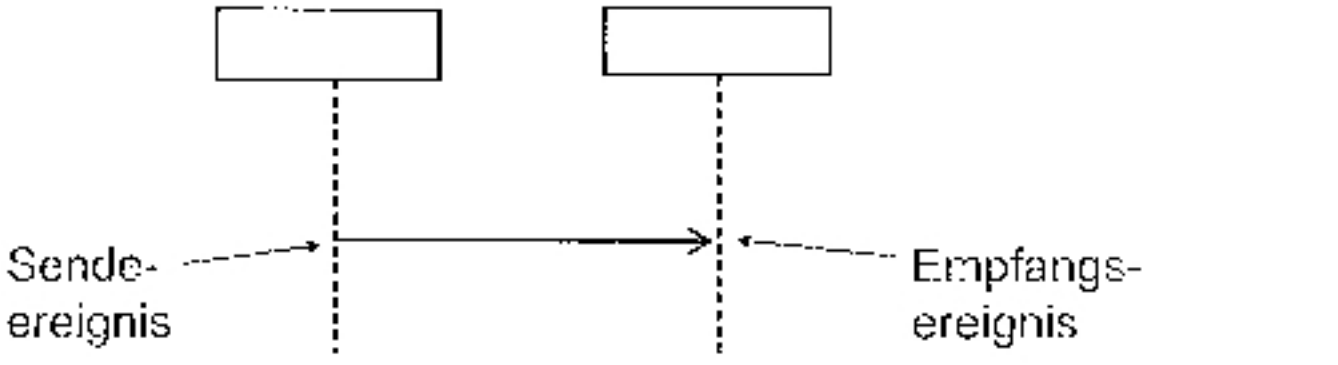
Notation



9.4.1 Ereignisspezifikation

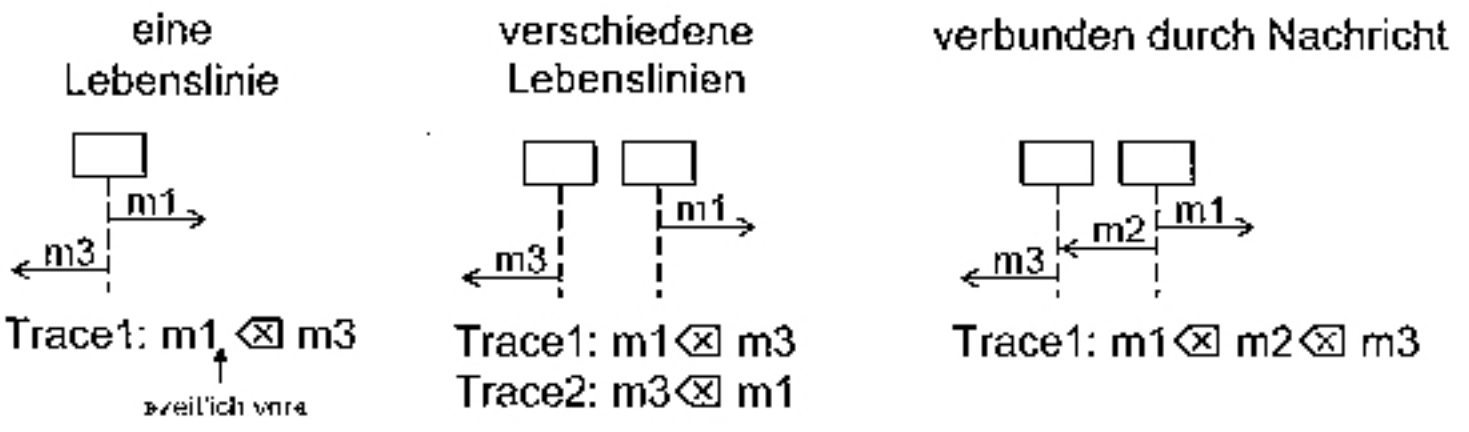
- Interaktionen werden als **Folge von Ereignisspezifikationen** auf Lebenslinien betrachtet

- Beispiel für Ereignisspezifikationen
  - **Senden** und **Empfangen** von Nachrichten auf verschiedenen Lebenslinien oder der gleichen Lebenslinie



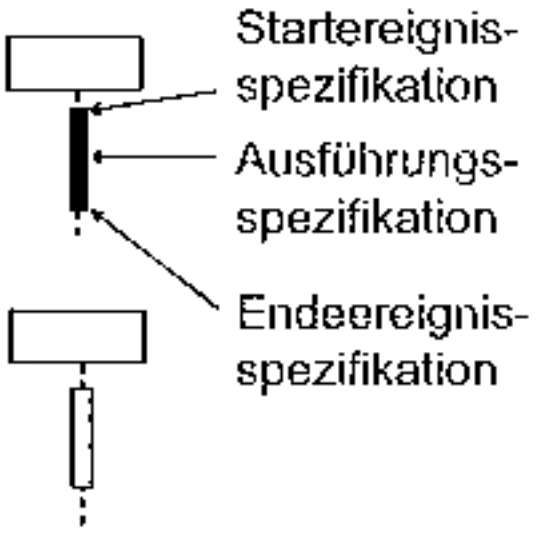
9.4.2 Reihenfolge

- Reihenfolge von Ereignisspezifikationen
  - **Vertikale Zeitachse** bestimmt nur die **Ordnung** von Ereigniseintritten **pro Lebenslinie**
    - Jedoch **nicht** die Reihenfolge von Ereigniseintritten **auf verschiedenen Lebenslinien**
  - Erst durch **Nachrichten zwischen Lebenslinien** wird eine Ordnung über Lebenslinien hinweg erzwungen



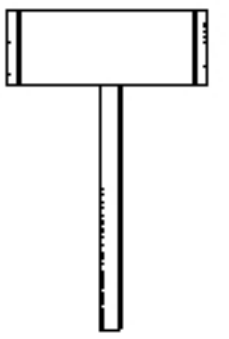
9.4.3 Ausführungsspezifikation

- Die **Ausführung** einer Aktivität/Operation wird durch zwei Ereignisspezifikationen (Start und Ende) auf der gleichen Lebenslinie definiert
- Diese sogenannte **Ausführungsspezifikation** kann durch einen Balken dargestellt werden
- Ausführungsarten
  - **Direkt**
    - Interaktionspartner führt Verhalten selbst aus
  - **Indirekt**
    - Ausführung wird an andere Interaktionspartner delegiert



9.4.4 Aktives Objekt

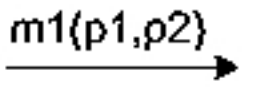
- **Aktive Objekte** verfügen über **eigenen Kontrollfluss** (Prozess oder Thread)
- Können **unabhängig** von anderen Objekten operieren
- **Notation**
  - Kopf der Lebenslinie wird links und rechts mit **doppeltem Rand** versehen
  - **durchgehender** Balken über gesamte Lebenslinie



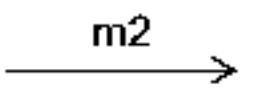
9.5 Nachricht

- Arten der Nachrichtenübermittlung

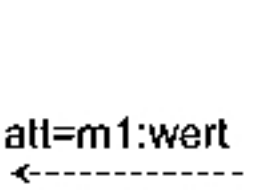
- **Synchroner Kontrollfluss**
  - Der Sender wartet bis zur Beendigung der Interaktion, die durch die Nachricht ausgelöst wurde



- **Asynchroner Kontrollfluss**
  - Die Nachricht wird als Signal betrachtet
  - Der Sender **wartet nicht** auf das Ende der Interaktion

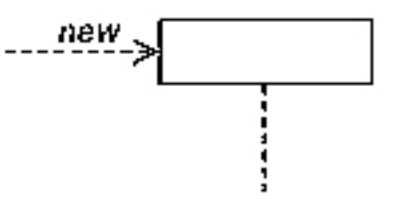


- **Antwortnachricht** (optional)

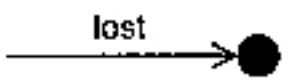


9.5.1 Spezielle Nachrichtenarten

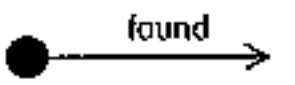
- **Objekterzeugung**
  - Ermöglicht, einen Interaktions-partner erst im Laufe der Interaktion zu erzeugen



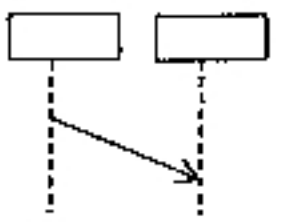
- **Verlorene** Nachricht
  - Senden einer Nachricht an unbekannten oder nichtrelevanten Interaktionspartner



- **Gefundene** Nachricht
  - Empfang einer Nachricht von einem unbekannten oder nicht relevantenInteraktionspartner

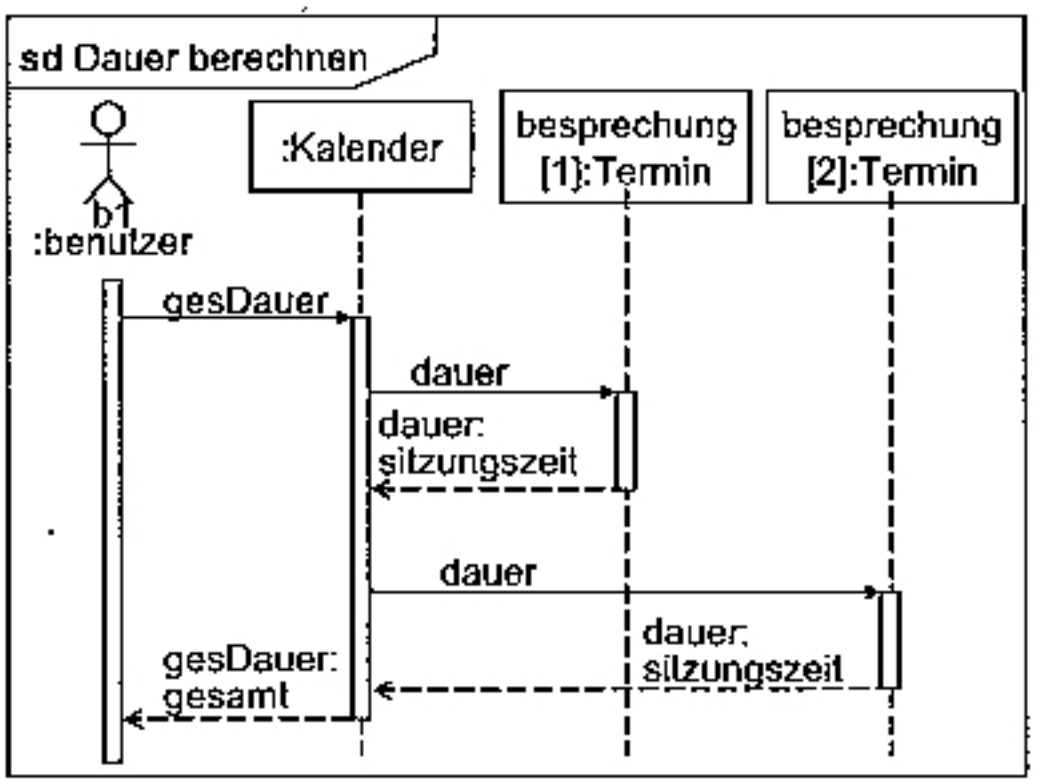


- **Zeitkonsumierende** Übertragung



9.5.2 Beispiel

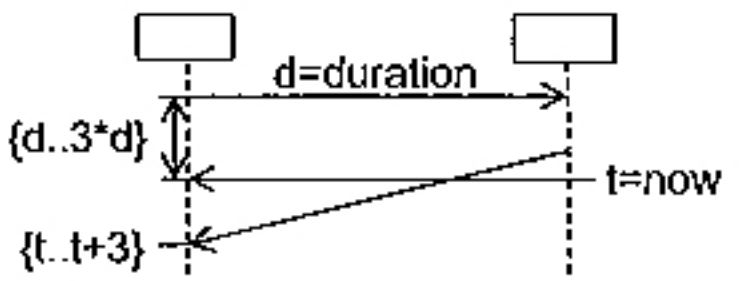
- Berechnung der Dauer eines Termins



9.5.3 Zeiteinschränkungen

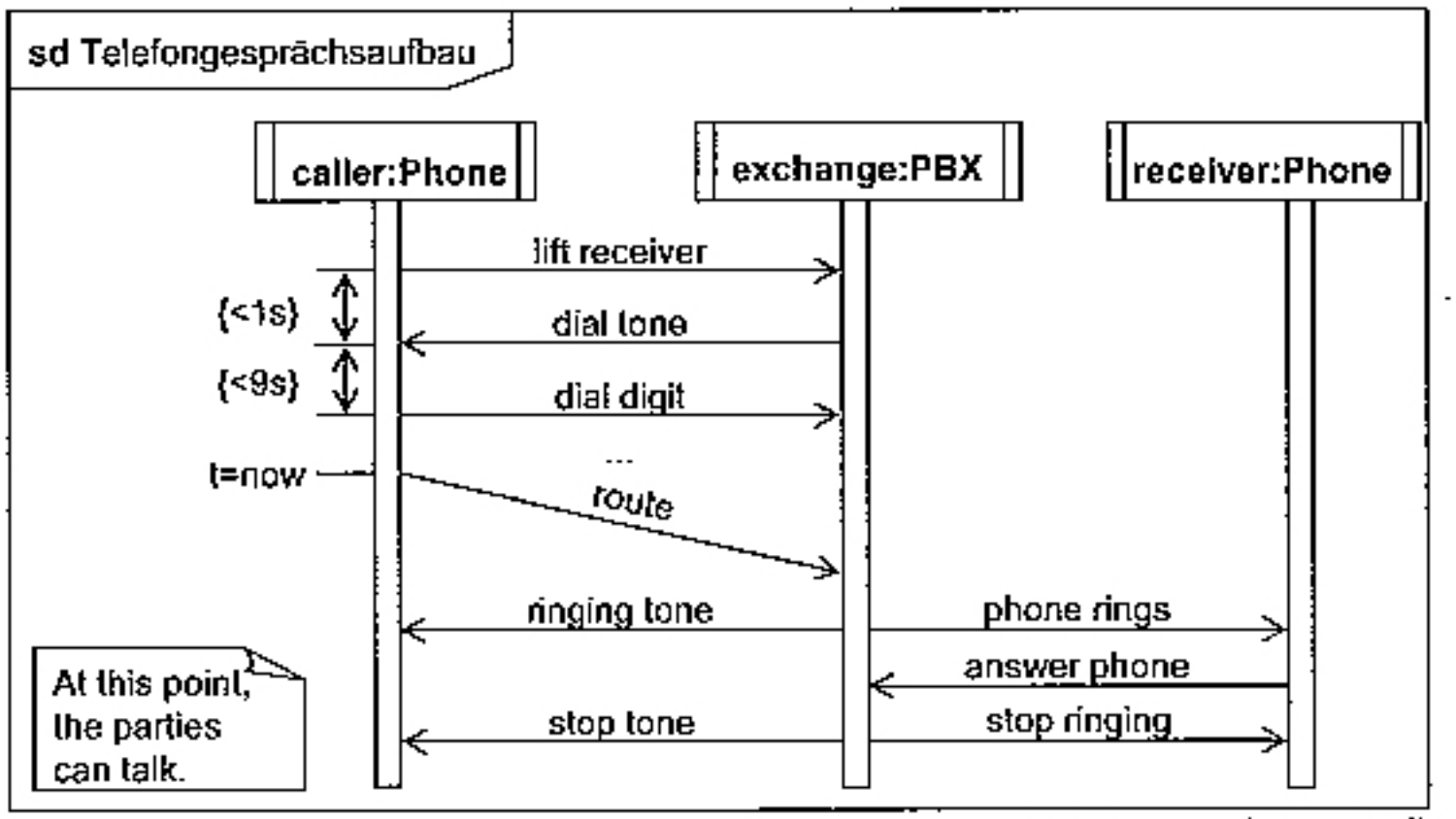
- Arten
  - **Zeitpunkt** (time constraint)
    - Bezieht sich auf einzelne Ereignisspezifikation
  - **Zeitdauer** (duration constraint)
    - Bezieht sich auf Zeitintervall zwischen zwei Ereignisauftritten

- Vordefinierte Aktionen zur Zeitberechnung
  - **now**: Berechnung der **aktuellen Zeit**
  - **duration**: Berechnung einer **Zeitdauer**
  - Erhaltene Werte können Variablen zugewiesen werden
  - Variablen können in Zeitausdrücken verwendet werden



9.5.4 Beispiel

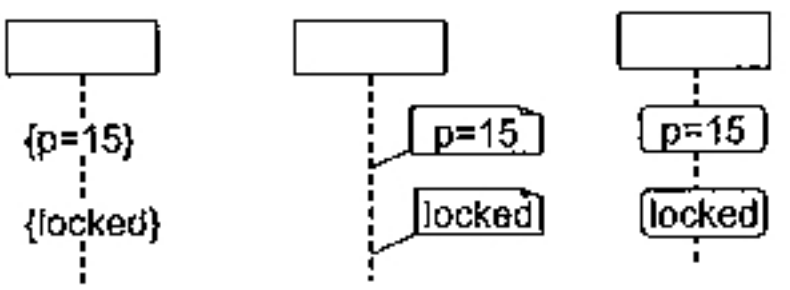
- Aufbau eines Telefongesprächs



(basierend auf [Rumb05])

9.6 Zustandsinvariante

- **Zusicherung**, dass eine **bestimmte Bedingung** zu einem **bestimmten Zeitpunkt** erfüllt ist
- Bezieht sich immer auf eine **bestimmte Lebenslinie**
- Wird **vor Eintritt** des darauf folgenden **Ereignisses ausgewertet**
- Falls Zustandsinvariante **nicht erfüllt** ist - **Fehler**
- Notationsvarianten

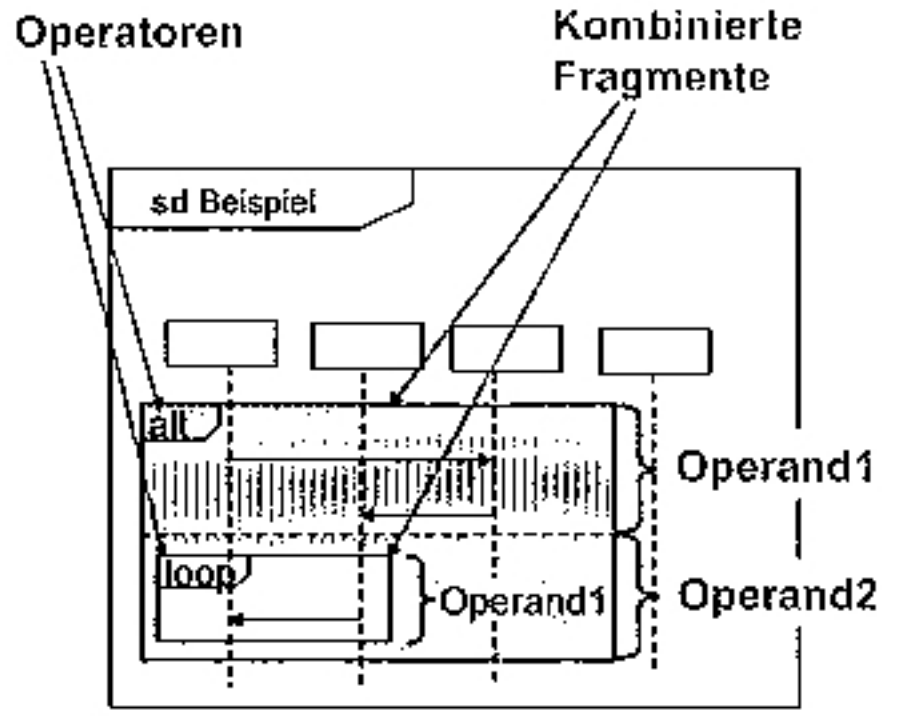


9.7 Kombinierte Fragmente

- Modellierung von **Kontrollstrukturen**
- Bestandteile: **Operator** und **Operanden**
- **Operator**
  - Definiert Art des kombinierten Fragments
  - 12 vordefinierte Operatoren
- **Operand**
  - Ein Operator enthält 1 **oder mehrere Operanden**, je nach Operatorart
  - Kann **Interaktionen, kombinierte Fragmente** (Schachtelung!) und **Referenzen auf Sequenzdiagramme** umfassen

9.7.1 Notation

- Kombiniertes Fragment wird wie Sequenzdiagramm mit Rahmen dargestellt
- Art des Fragments wird durch Operator im Pentagon festgelegt (default: seq)
- Operanden werden durch gestrichelte Linien voneinander getrennt

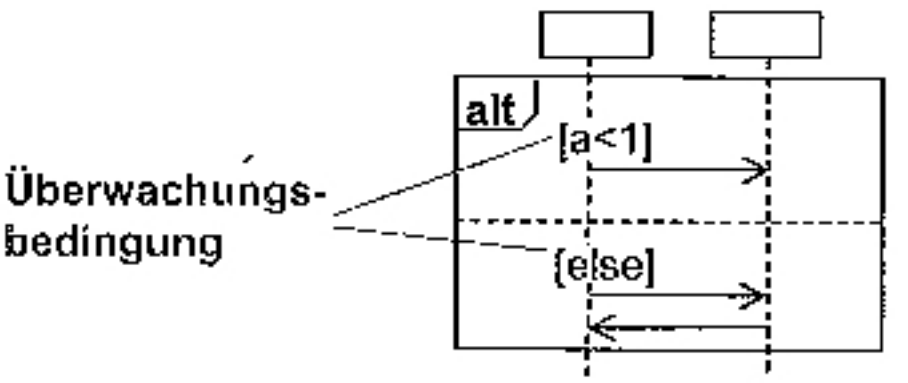


9.7.2 Operatorarten

	Operator	Zweck
Verzweigungen und Schleifen	**alt**	Alternative Interaktionen
	opt	Optionale Interaktionen
	break	Ausnahme Interaktionen
	loop	Iterative Interaktionen
Parallelität und Ordnung	**seq**	Sequentielle Interaktionen mit schwacher Ordnung (Default-Operator)
	strict	Sequentielle Interaktionen mit strenger Ordnung
	par	Parallele Interaktionen
	critical	Atomare Interaktionen
Filterungen und Zusicherungen	**ignore**	Irrelevante Interaktionen
	consider	Relevante Interaktionen
	assert	Zugesicherte Interaktionen
	neg	Ungültige Interaktionen

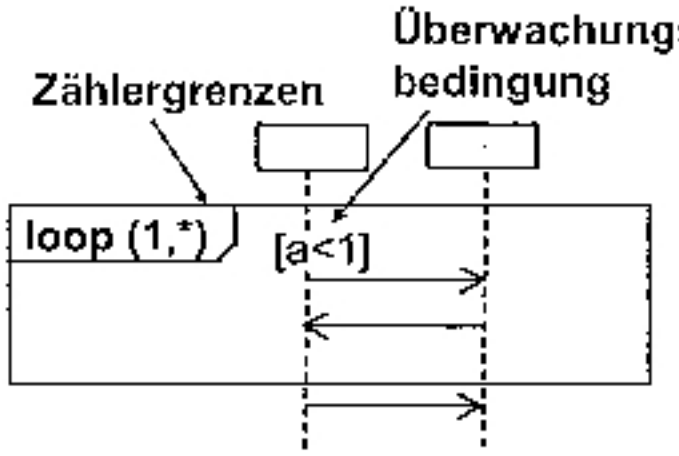
9.7.3 Verzweigungen und Schleifen: alt-Operator

- Darstellung von zwei oder mehreren **alternativen Interaktionsabläufen**
- Zur Laufzeit wird **maximal ein Operand** ausgeführt
- Auswahl eines Operanden anhand von **Überwachungsbedingungen**
  - **Boolscher Ausdruck** in **eckigen Klammern**
  - Vordefinierte **else-Bedingung**: Operand wird ausgeführt, falls die Bedingungen aller anderen Operanden nicht erfüllt sind



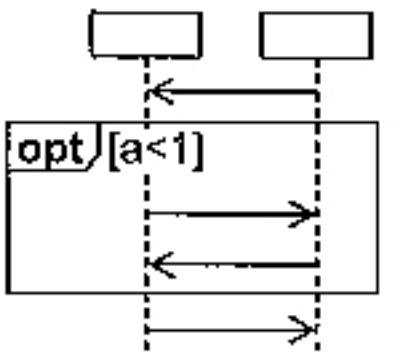
9.7.4 Verzweigungen und Schleifen: loop-Operator

- Darstellung einer **Schleife** über einen bestimmten Interaktionsablauf
- Fragment enthält nur **einen Operanden**
- Ausführungshäufigkeit wird durch **Zähler** mit Unter- und Obergrenze dargestellt
- Optional: **Überwachungsbedingung**, wird bei jedem Durchlauf überprüft

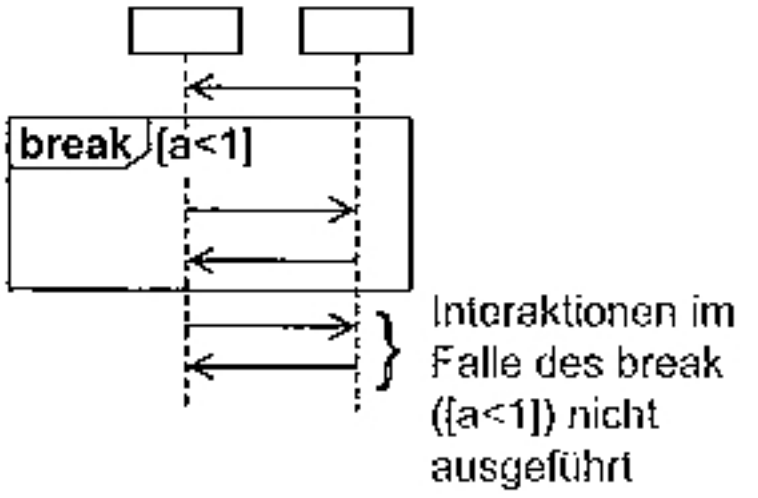


9.7.5 Verzweigungen und Schleifen: opt- und break-Operator

- Überwachungsbedingung steuert Durchlauf der Interaktionen
- Optionale Interaktionen



- Ausnahme-Interaktionen



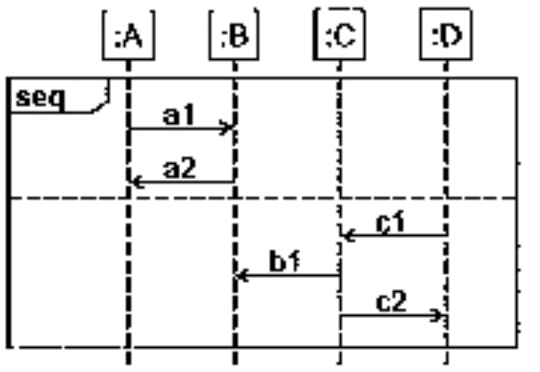
9.7.6 Parallelität und Ordnung: seq-Operator / strict-Operator

- Sequentielle Interaktion mit schwacher Ordnung

  Trace1: a1 ⌫ a2 ⌫ c1 ⌫ b1 ⌫ c2

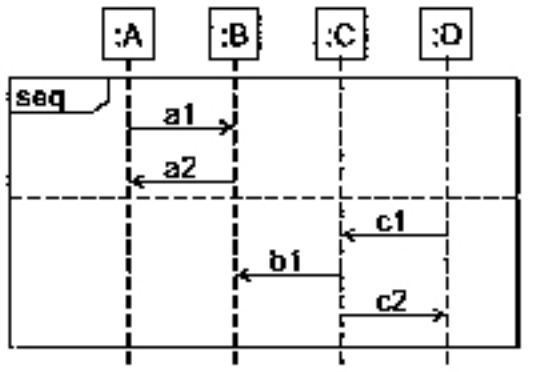
  Trace2: a1 ⌫ c1 ⌫ a2 ⌫ b1 ⌫ c2

  Trace3: c1 ⌫ a1 ⌫ a2 ⌫ b1 ⌫ c2



- Sequentielle Interaktion mit strenger Ordnung

  Trace1: a1 ⌫ a2 ⌫ c1 ⌫ b1 ⌫ c2



9.7.7 Parallelität und Ordnung: par-Operator

- Nebenläufige Interaktionen
  - Lokale Reihenfolge pro Operand muss erhalten bleiben - z.B. **a1 vor b1**

  Trace1: a1 ⌫ a2 ⌫ c1 ⌫ b1 ⌫ c2

  Trace2: a1 ⌫ c1 ⌫ a2 ⌫ b1 ⌫ c2

  Trace3: a1 ⌫ c1 ⌫ b1 ⌫ a2 ⌫ c2

  Trace4: a1 ⌫ c1 ⌫ b1 ⌫ c2 ⌫ a2

  Trace5: c1 ⌫ a1 ⌫ a2 ⌫ b1 ⌫ c2

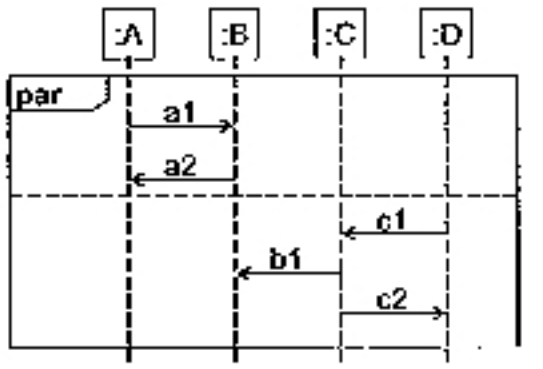
  Trace6: c1 ⌫ a1 ⌫ b1 ⌫ a2 ⌫ c2

  Trace7: c1 ⌫ a1 ⌫ b1 ⌫ c2 ⌫ a2

  Trace8: c1 ⌫ b1 ⌫ a1 ⌫ a2 ⌫ c2

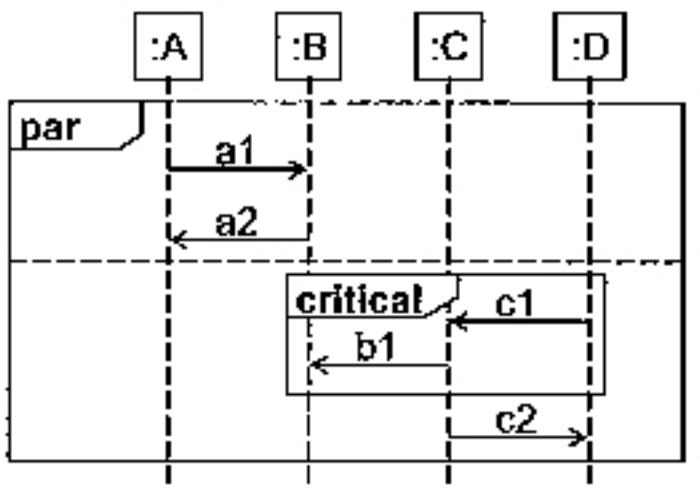
  Trace9: c1 ⌫ b1 ⌫ a1 ⌫ c2 ⌫ a2

  Trace10: c1 ⌫ b1 ⌫ c2 ⌫ a1 ⌫ a2



9.7.8 Parallelität und Ordnung: critical-Operator

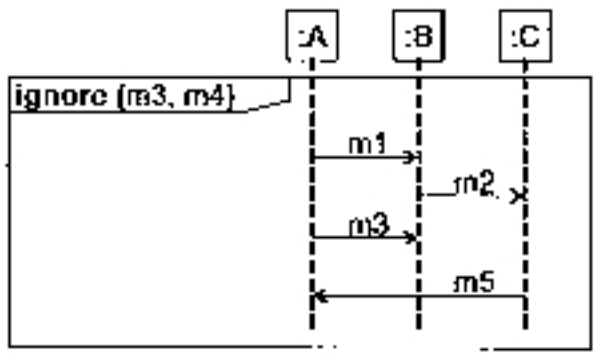
- Kritischer Bereich
  c1 unmittelbar vor b1



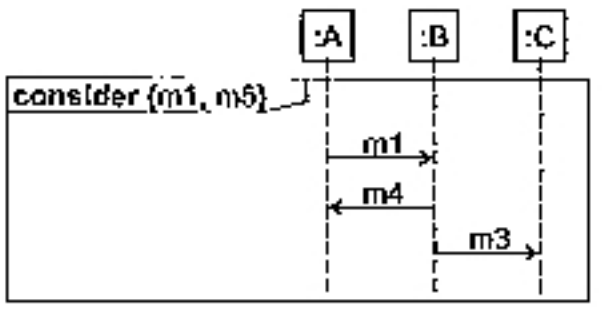
Trace1: a1 ⌫ a2 ⌫ c1 ⌫ b1 ⌫ c2
Trace2: a1 ⌫ c1 ⌫ b1 ⌫ a2 ⌫ c2
Trace3: a1 ⌫ c1 ⌫ b1 ⌫ c2 ⌫ a2
Trace4: c1 ⌫ b1 ⌫ a1 ⌫ a2 ⌫ c2
Trace5: c1 ⌫ b1 ⌫ a1 ⌫ c2 ⌫ a2
Trace6: c1 ⌫ b1 ⌫ c2 ⌫ a1 ⌫ a2

9.7.9 Filterungen und Zusicherungen: ignore-Operator / consider-Operator

- Irrelevante Interaktionen

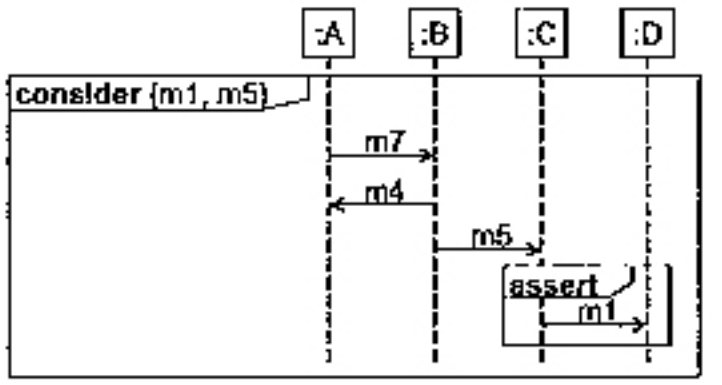


- Relevante Interaktionen

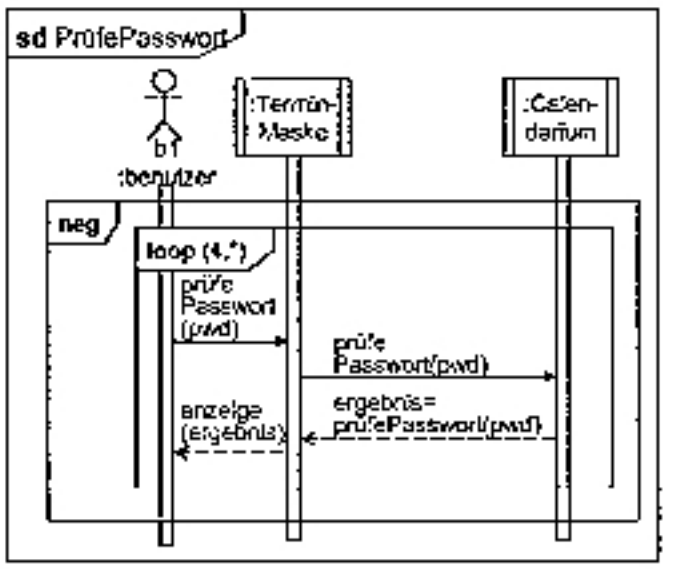


9.7.10 Filterungen und Zusicherungen: assert-Operator / neg-Operator

- Zugesicherte Interaktionen



- Ungültige Interaktionen



9.8 Modularisierung

- **Zweck**: Wiederverwendung und Reduktion der Komplexität

- **Interaktionsreferenz**
  - Zur Referenzierung anderer Sequenzdiagramme
  - Dadurch können Interaktionsabläufe und Lebenslinien zerlegt werden

- **Fortsetzungsmarke**
  - Zur Zerlegung der Operanden eines **alt**-Operators

- **Verknüpfungspunkt**
  - Zur Verbindung von Nachrichten zwischen Sequenzdiagrammen, Interaktionsreferenzen oder kombinierten Fragmenten

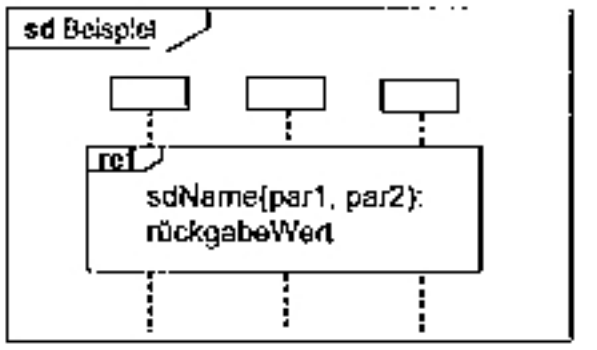
9.8.1 Interaktionsreferenz

- **Interaktionen** des referenzierten Sequenzdiagramms werden **ausgeführt**, so als wären sie direkt in das referenzierende Diagramm **eingebettet**

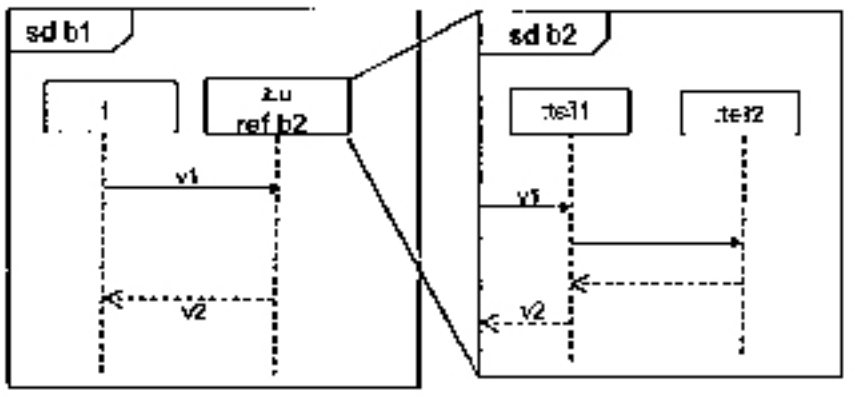
- Eventuell vorhandene **Parameter** werden gebunden

- Nach Ausführung der referenzierten Interaktionen wird **unterhalb der Interaktionsreferenz fortgesetzt**

- Zerlegung von Interaktionsabläufen
  - Rahmen mit Pentagon in der linken oberen Ecke
  - Pentagon enthält Schlüsselwort **ref**
  - Rahmen enthält Namen der referenzierten Interaktion, optional Parameter und Rückgabewerte



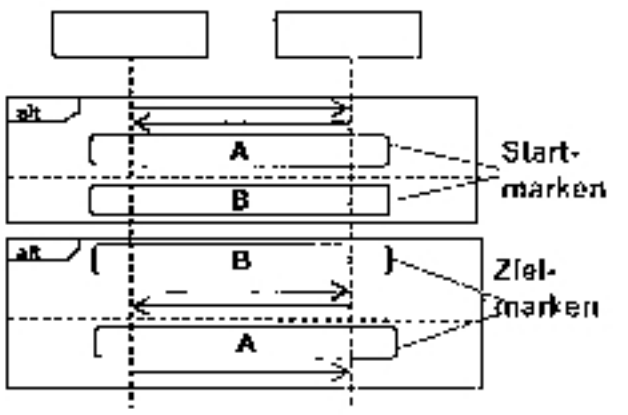
- **Zerlegung von Lebenslinien**
  - **Lebenslinien** können **interne Strukturen** aufweisen, für die eigene Sequenzdiagramme spezifiziert werden können
  - Schlüsselwort **ref** im Kopf der Lebenslinie



9.8.2 Fortsetzungsmarke

- **Zweck**: Zerlegung der Operanden eines alt-Fragments

- **Startmarke** am Ende eines Interaktionsteils verweist auf **Zielmarke** am Beginn eines anderen Interaktionsteils
  - Gleiche Markennamen
  - Überdeckung gleicher Menge an Lebenslinien
  - Rechtecke mitabgerundeten Ecken



- Erreicht die Ausführung die Startmarke, wird mit den Interaktionen unterhalb der Zielmarke fortgefahren
  - **Achtung**: kein Rücksprung zur Startmarke möglich!

9.8.3 Verknüpfungspunkt

- **Zweck**: ermöglicht den **Nachrichtenfluss zwischen**
  - Sequenzdiagrammen und/oder
  - kombinierten Fragmenten und/oder
  - Interaktionsreferenzen

- Dadurch können **hereinkommende** oder **hinausgehende Nachrichten** modelliert werden

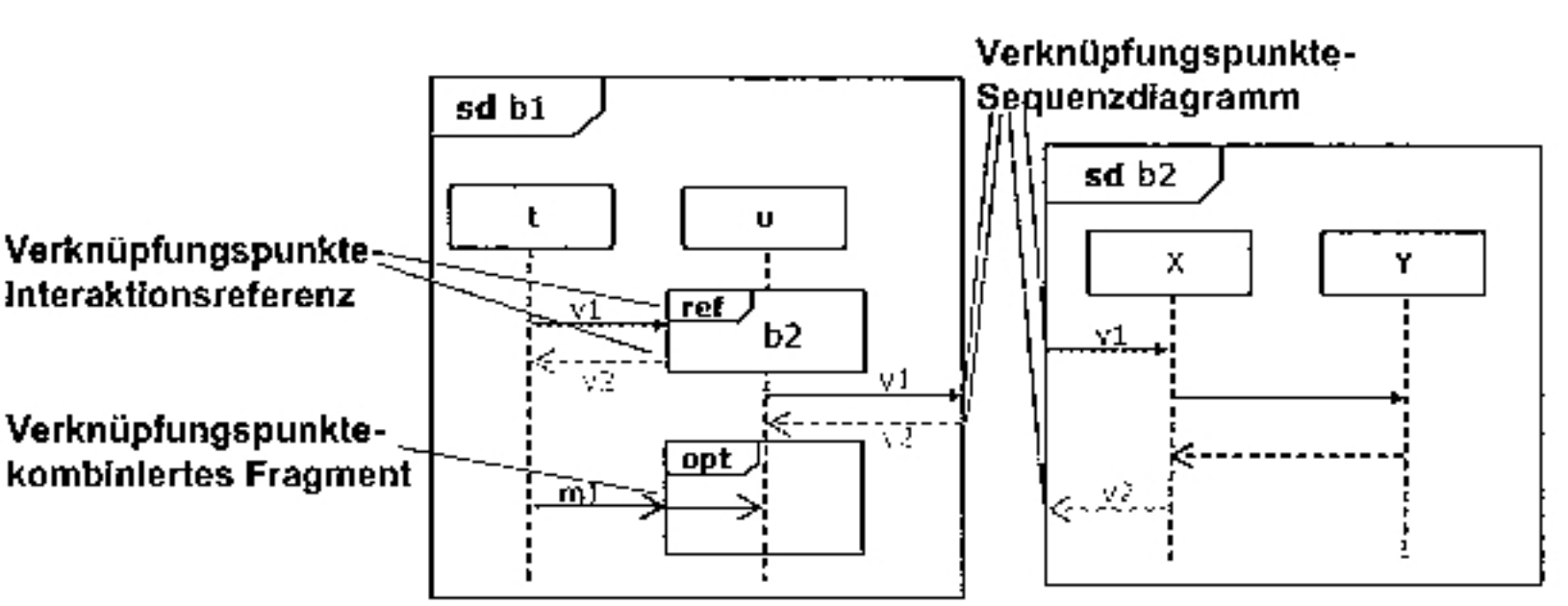
- Verknüpfungspunkte zwischen Sequenzdiagrammen können
  - entweder implizit über Namensgleichheit
  - oder explizit über eine Interaktionsreferenz

  modelliert werden

Verknüpfungspunkt

- Spitze oder Ende eines Nachrichtenpfeils berührt Rand des Rahmes

- Optional: Name für Verknüpfungspunkt wird neben Schnittpunkt von Nachrichtenpfeil und Rahmen angegeben

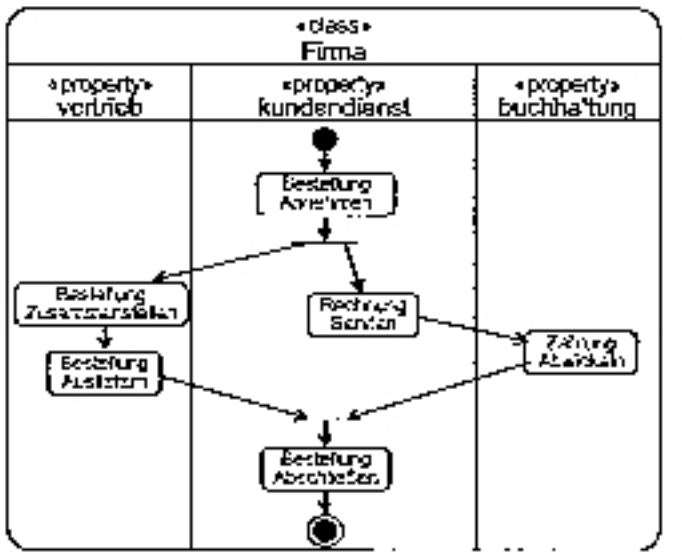


9.9 Exkurs: Sequenzdiagramm vs. Aktivitätsdiagramm

- Pfeil im Aktivitätsdiagramm
  - Zeigt Abhängigkeiten zwischen Aktionen
  - Zeigt NICHT: Nachrichten zwischen Objekten
  - Entspricht einer, mehreren oder keiner Nachricht

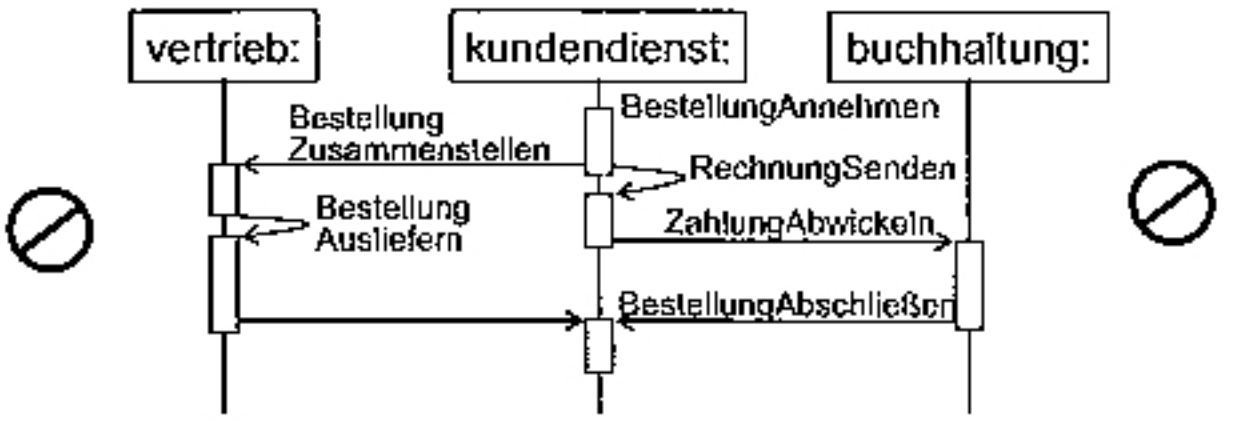
- Pfeil im Sequenzdiagramm
  - Zeigt Nachrichten zwischen Objekten
  - Entspricht Operation einer Klasse

- Beispiel: Akitivitätsdiagramm



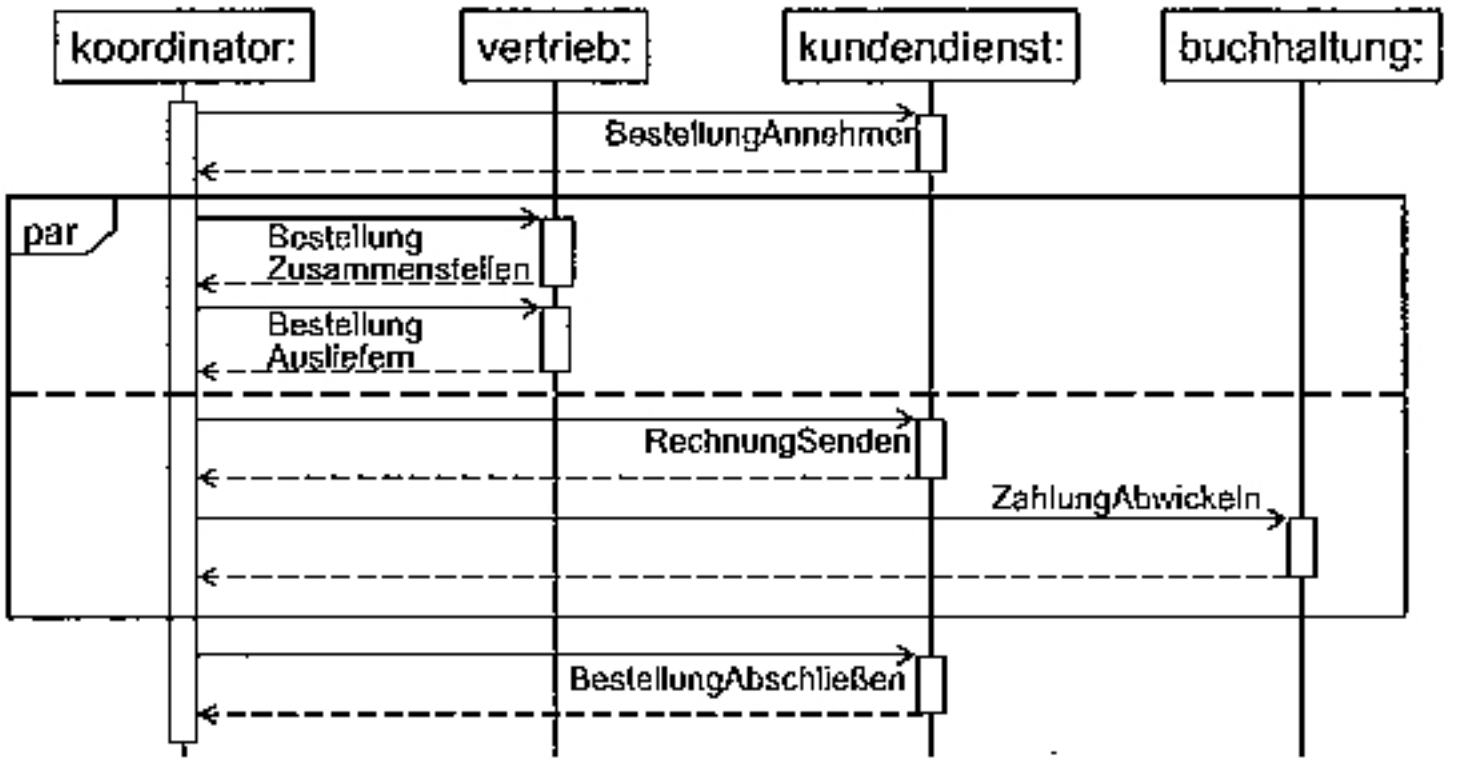
Exkurs: Sequenzdiagramm vs. Aktivitätsdiagramm

- Beispiel: Sequenzdiagramm – **falsche Transformation**
  - Pfeile können nicht zu Nachrichten transformiert werden
  - Beispielsweise kann RechnungSenden nicht das Abschicken der Nachricht ZahlungAbwickeln beinhalten
  - RechnungSenden ist zu Ende, bevor ZahlungAbwickeln gestartet wird
  - Damit wird sichergestellt, dass RechnungSenden wiederverwendbar ist, ohne festzulegen, was vorher und nachher passieren muss



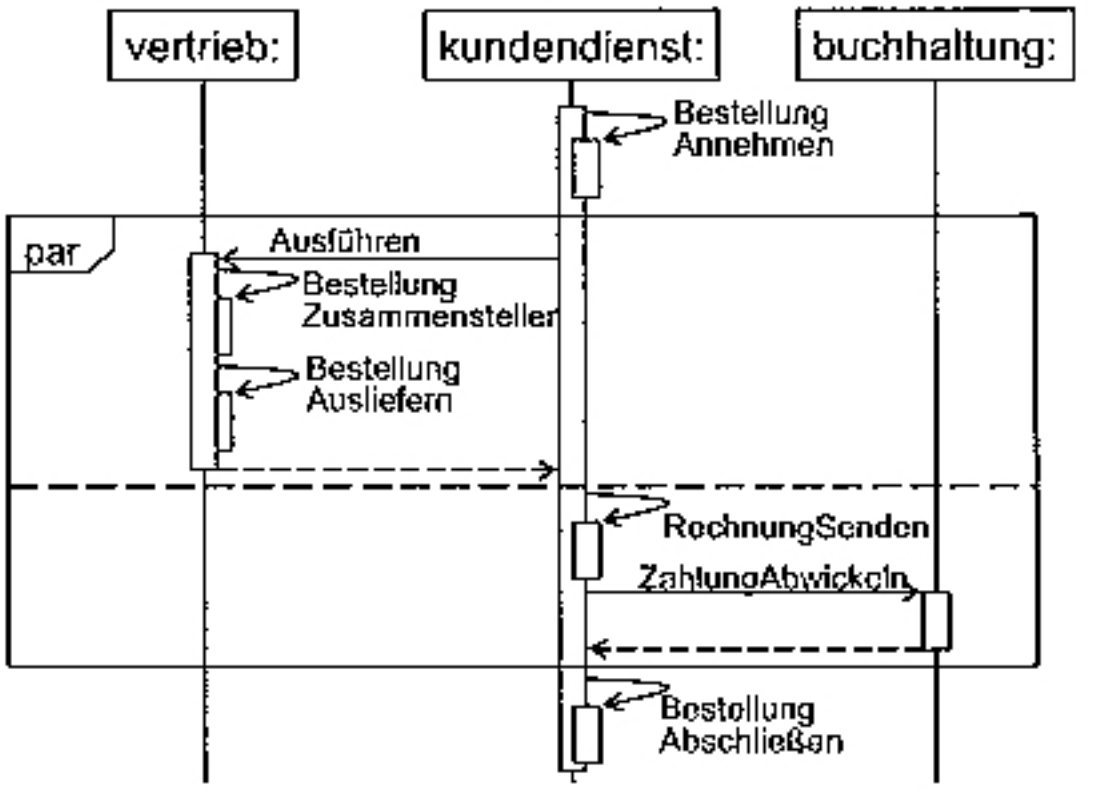
Exkurs: Sequenzdiagramm vs. Aktivitätsdiagramm

- Beispiel: Sequenzdiagramm – Alternative 1



Exkurs: Sequenzdiagramm vs. Aktivitätsdiagramm

- Beispiel: Sequenzdiagramm – Alternative 2

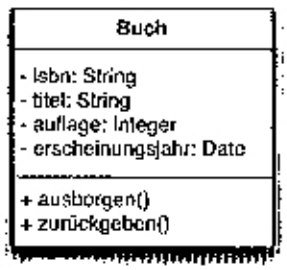


Literatur und Quellen:

- Kapitel 4.4 in:
  Martin Hitz, Gerti Kappel, Elisabeth Kapsammer, Werner Retschitzegger:
  UML @ Work - Objektorientierte Modellierung mit UML2.
  dpunkt Verlag 2005 / 3. aktualis. u. überarb. Aufl. 2005.
  ISBN-13: 9783898642613 ISBN-10: 3898642615

6.1.1 Klassen

- Klassen werden als Rechteck dargestellt, das horizontal in einzelne Abschnitte untergliedert ist.
- Mit Ausnahme des obersten Abschnitts können alle Abschnitte fehlen, wobei das Nichtaufscheinen eines Abschnitts im Diagramm keine Rückschlüsse auf die Existenz der durch ihn verwalteten Charakteristika zulässt.



- Im obersten Abschnitt sind **Name und allgemeine Eigenschaften** der Klasse vermerkt.
- Der zweite Abschnitt enthält die **Attribute**.
- Der dritte Abschnitt enthält die **Operationen**.
- Danach sind prinzipiell noch weitere Abschnitte zulässig, die jedoch nicht von UML normiert wurden.

Erster Abschnitt

- Im ersten Abschnitt können neben dem Klassennamen noch ein **Stereotyp** und Eigenschaftsangaben eingetragen werden.
- Stereotypen sind Erweiterungsmechanismen, mit denen die Klassenhierarchie des Metamodells erweitert werden kann, indem jedes Modellelement (z.B. Klasse) mit höchstens einem Stereotyp weiter kategorisiert werden kann.
- Stereotype erscheinen in den "stereotypischen" Anführungszeichen << ... >> oberhalb des Klassennamens.
- Eigenschaftsangaben erfolgen durch Auflistung in geschwungenen Klammern { ... }. Eine häufig benützte Eigenschaft ist **abstract** zur Kennzeichnung einer abstrakten Klasse. Alternativ kann eine abstrakte Klasse durch einen kursiv geschriebenen Namen dargestellt werden.